Business Rules based Legacy System Evolution towards Service-Oriented Architecture

Ph.D. Thesis

Yang Xu

Software Technology Research Laboratory

Faculty of Computing Sciences and Engineering

De Montfort University

2010

To my fiancee and
my parents.

Declaration

I declare that the work described in this thesis was originally carried out by me during the period of registration for the degree of Doctor of Philosophy at De Montfort University, U.K., from 2003 to 2010. It is submitted for the degree of Doctor of philosophy at De Montfort University. Apart from the degree that this thesis is currently applying for, no other academic degree or award was applied for by me based on this work.

Acknowledgements

For many years I had been dreaming about receiving a PhD. I would like to thank many people who helped me in achieving this dream in different ways when I undertook the work of this thesis.

I wish to express my most profound thanks to my supervisors, Prof. Hongji Yang and Dr. Issam Amin for their invaluable advice, experienced guidance and encouragement during my four-year study. They provided me with many useful comments and suggestions for preparation of this thesis.

My thanks must go to Prof. Hussein Zedan, the director of the laboratory, for providing many helpful suggestions during our many discussions. My research career benefits tremendously from the research methodologies that Prof. Hussein Zedan introduced to me.

I would also like to thank the Research Office at De Montfort University for their outstanding management.

I wish to express thanks to my fiancee, Judy Chiang, my parents for all their loves, encouragements, patience and supports over the years. This thesis is dedicated to them.

Finally, I would like to thank all of the colleagues in Software Technology Research Laboratory at De Montfort University, for their valuable suggestions and discussions, for their encouragement and support, for building great software packages, including: Feng Chen, Han Li, Jianzhi Li, Shaoyun Li, Zhuopeng Zhang, and many others. I am indebted to all of them.

Publications by Author

Y. Xu, H. Yang and I. Amin, "Business Rule Based Program Transformation for CRM System Evolution," in Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2006), USA, 2006, pp. 244-247.

Y. Xu, Q. Duan and H. Yang, "Web Services Oriented Customer Relationship Management System (CRM) Evolution," in Proceedings of the 2005 IEEE International Workshop on Software Technology and Engineering Practice (STEP 2005), Hungary, 2005, pp. 39-48.

J. Pu, Z. Zhang, Y. Xu and H. Yang, "Reusing Legacy COBOL Code with UML Collaboration Diagrams via a Wide Spectrum Language," in Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration (IRI'05), IEEE Systems, Man, and Cybernetics Society, 2005, pp. 78-83.

J. Pu, Z. Zhang, R. Millham, Y. Xu and H. Yang, "Modelling Web-Based System with UML Sequence Diagrams," in Proceedings of IADIS Virtual Multi Conference on Computer Science and Information Systems (MCCSIS'05), International Association for Development of the Information Society, 2005.

Y. Xu and H. Yang, "Data Analysis for Customer Relationship Management," in Proceedings of the 10th Chinese Automation and Computing Society Conference in the UK (CACSUK), Liverpool, England, 2004, PP. 227-232.

Abstract

This is a critical time for companies to bring technology and business together as never before. An accelerating competition demands businesses change at a pace that was inconceivable a decade ago. Business must respond with ever faster continuous improvement of existing operations and the constant introduction of new products. Only companies that master the required rhythm of change will persevere and prosper. Global collaboration and emerging corporate cultures are creating a new type of innovative enterprise: one based on services.

Enterprises can be empowered to live up to the potential of becoming dynamic, agile and real-time. Service orientation is emerging from the amalgamation of a number of key business, technology and cultural developments. Three essential trends in particular are coming together to create a new revolutionary breed of enterprise, the service-oriented enterprise (SOE): (1) the continuous performance management of the enterprise; (2) the emergence of business process management; and (3) advances in the standards-based service-oriented infrastructures.

This thesis focuses on this emerging three-layered architecture that builds on a service-oriented architecture framework, with a process layer that brings technology and business together, and a corporate performance layer that continually monitors and improves the performance indicators of global enterprises provides a novel framework for the business context in which to apply the important technical idea of service orientation and moves it from being an interesting tool for engineers to a vehicle for business managers to fundamentally improve their businesses.

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

List of Lists

Chapter 1

Introduction

1.1 Overview

People in the 21st century have embarked on an exciting era of implementing information technology into practical software development. This contemporary paradigm for developing software applications is rapidly gaining momentum in the business community. Previously, information systems built using old software development paradigms focused almost exclusively on internal and individual business processes. Nowadays, information systems built using emerging software development paradigm are progressively emphasising on cross-organisational business processes. Coupled with web service and service-oriented architecture techniques, the novel information systems are more like an aggregation of independent business applications that are flexible and relatively easy to maintain, upgrade and integrate. As a result, this up-and-coming breed of information system architecture brings a possibility for adapting highly competitive business models.

A wide majority of people are convinced that application development and integration have evolved from the art of writing code to the art of assembling a set of services into new business capabilities. Companies are finding themselves to be strategically disadvantaged if they do not make use of the new technology. However, while trying to move to the new arena, many large-scale companies are hindered by their heritage of stovepipe enterprise information systems, which are typically large, heterogeneous and mission-critical systems. It is commonplace that these inherited enterprise systems are strongly intertwined with existing business processes and policies. Although this allows efficient operations, it severely hinders their alignment with new business requirements. The maintenance and replacement of these systems, most of which have become legacy systems with the passage of time, has proved to be a complicated, time-consuming, expensive and risky effort. These systems are tightly

coupled to the business workflow and suffer from a rigid and contrived architecture which results from many years of ad hoc patching and fixing. They can offer very limited extension space to any further development.

Given such a situation, organisations are facing an unenviable dilemma. On the one hand, they want to get rid of their legacy systems as soon as possible in order to be able to align their information systems with new business processes. On the other hand, legacy applications and repository systems are considered important enterprise assets, harbouring significant investments in enterprise data and policies already at hand. Business processes often critically depend on the permanent availability of these legacy systems. Hence, one cannot annihilate the investments and simply replace legacy systems with new information systems. The key lies in the discovery and application of what are called business rules.

A business rule is a compact and simple statement that defines or constrains one aspect of business and intends to assert business structure or influence the behavior of business. Business rules make abstract business strategy concrete to govern business operation. Business strategy is designed to meet user requirements. When users change their requirements business rules have to be changed accordingly. By capturing the business rules companies will gain the ability to create systems fully aligned with their specific business needs.

This thesis introduces a novel framework which has technological innovation designed to help legacy system evolution. The thesis provides a thorough introduction to business rules, as well as a practical approach for extracting and integrating them into target systems. It shows how to identify and express business rules, offers practical strategies for their use, and explains the key elements of logic that underpin their application. The business rules migration is facilitated by a sophisticated process of comparison and adaptation to ensure the business rules and target systems are fit. The technology is appropriate for any kind of organisation, and is especially well suited for development in the rapidly expanding Web Services industry.

In brief, the Business Rule Approach represents a major paradigm shift in system design and development. Employing a business-driven approach, the use of business

rules has been proven to enhance the effectiveness, flexibility, and efficiency of business systems.

1.2 Motivation

Change is inherent in software. Whether programmers develop a new system or maintain an old one, software is in a constant flux. In this rapidly evolving world, an information system that needs to interact with its environment is inevitably required to evolve in order to keep its business competitiveness with others.

In the past few years, several strategies have been implemented in legacy systems evolution. The four most prominent ones are elimination, encapsulation, modernisation and maintenance. These strategies have different target customers respectively. The elimination strategy is suitable for those simple and unimportant legacy systems that product low business value. The encapsulation strategy (also known as wrapper) is the strategy of choice when legacy systems hold valuable and up-to-date assets that should be leveraged in the next generation of enterprise information systems. The modernisation strategy is preferably applied on the legacy systems that capture substantial business value but suffering from a poor technical condition. Finally, the normal maintenance strategy, sometimes referred to as gradual migration approach, is best for the legacy systems that can generate high business value and operating as technically sound systems. The elimination and maintenance strategies usually apply to simple tasks that only need to work on one information system. On the contrary, the other two strategies involve the work on more than one system or architecture. This definitely requires software engineers to put more energy to guarantee the retained legacy assets are compatible with new technology.

Despite the fact that these strategies have been used successfully in some situations, they still suffer from several major drawbacks:

1. All these existing evolution strategies revolve around ad hoc patching from software engineering perspective, but rarely consider the consistency of business activities. These approaches are typically oriented toward realising technical integration between new technology and legacy systems with virtually

no regard for the new business process requirements that the legacy systems are supposed to facilitate.

2. Legacy assets cannot be re-used adequately. Existing evolution strategies typically based on the assumption of new processes. New developed system make use of legacy systems with minor adjustments to the legacy data and functionality to make them accessible to external resources. This assumption is not realistic because legacy systems are geriatric systems that implement partly outdated business processes and policies. In addition, their old fashion nature makes them inappropriate building blocks for composition into consistent and cohesive business processes.

3. The encapsulate strategy usually seek to integrate legacy systems with new business applications in a purely bottom-up manner without taking into account alignment with new top-down engineered business process requirements. This method tends to exploit programming logic and data that are encapsulated in legacy systems, largely neglecting requirements imposed by new business processes in the business domain.

4. Finally, any effective approach to align new business requirements with legacy systems must be methodology-oriented. A symmetric methodology that combines bottom-up reverse engineering of legacy systems with top-down design of business processes is required.

All in all, most of the existing strategies engage focus on solving problems from a software engineering perspective rather than business perspective. This is believed to provide insufficient guidance on maintaining business knowledge and processes. Business professionals should be involved in maintaining business knowledge and expressing business needs in very direct, concrete ways. Adopting these principles in practice will lead to a new approach that profoundly affects the roles of both business professionals and IT professionals. Therefore, to enable a clear understanding of legacy system and raise the success rate of system evolution, a more defined method which gives consideration to both business and technology aspects is required.

1.3 Aims and Objectives

The proposed Service-Oriented Architectures (SOAs) are providing unparalleled integration within and between enterprises. Performance monitoring and management are delivering incredible visibility to business practices. But what is even more exciting is the bridge between technology and business through automated business processes. IT and businesses professions are involved in continuous improvement feedback loops. Automated business processes can improve that feedback mechanism and thus can keep the IT and business goals better in sync. So sandwiched between the technical service-oriented architectures and the business-focused performance management solution trends are the emerging business process management platforms, which are automating business policies and procedures and are supporting better business – IT alignment with continuous improvement of business process implementations.

New requirements are always the driver of legacy systems evolution. In general terms, people involved in this process can be categorised into two groups: system users and system developers. The significant gaps between these two communities often cause problems in legacy system maintenance and evolution. This thesis aims to help users and developers come closer in the process of information systems development and operation, specifically, to integrate legacy business knowledge into target information systems and enable business people directly configure new systems at run-time.

The objectives cover a wide range of topics. To start, the work has to be phased over a number of steps. Similarly, in each step a specific objective is assigned. Thus, the main objectives of this thesis can be recognised as a list of concrete and smaller objectives as below:

- To help software engineers gain a deep understanding of legacy systems.
- To provide a systematic way of capturing, documenting and retaining the legacy business knowledge.
- To lower the technical requirement for business professionals managing

business rules.

- To enable software engineers to make business rules as a comfortable part of the target systems.
- To explain how technology and business objectives influence the software evolution processes.

Reflecting to these objectives, the desired target information system should obtain the following features:

- Business rules management is supported as a stand-alone component. In this case, the business can acquire a best-fit or best-of-breed business rules management package and incorporate it into its computing architecture as appropriate.
- A graphical user interface (business rule composer), in which business professionals and IT professionals can use in various ways to efficiently develop and apply rules and policies.
- Coupling with a workflow. This powerful combination features automated process management with coordination and decision making provided by the business rules technology. The result permits not only more sophisticated workflow control but automation of selected decision-making tasks as well. A seamless integrated framework, which enables applications to invoke a business policy or a set of business rules via a standard format and protocol.
- Including a rich set of Application Programming Interfaces (APIs) for other packaged software bundle into target information systems. Many other kinds of software capabilities, such as customer relationship management (CRM) and enterprise resource planning (ERP), provide natural niches for business logic technology, which will undoubtedly be exploited to the full extent their architectures permit.

The cooperation of these features leads to better cooperation between business professionals and IT professionals, reduced implementation times and fewer

opportunities for interpretation errors, eventually, keeps the business goals of target systems in an advantageous position in an ever-increasing competitive and complex business environment.

Objectives for business professionals: This thesis will introduce a new service-focused approach of doing business as the modus operandi. It will explain how technology and business objectives influence the software modernisation processes. By learning the proposed approach, business professionals can rethink their roles in the system development phase. Furthermore, this thesis can also teach them a new way to operate information systems in a flexible and simple nature.

Objectives for IT professionals: This thesis will introduce a standard and effective way of connecting businesses, explain the reason that legacy system evolution have failed or succeeded in practice. It will also provide insight into how developers can repeat these successes and avoid the failures. This thesis will help IT professionals understand the impact of legacy code, coupled with incremental development and deployment practices, on design activities.

The thesis will focus upon proven methods and tools for:

- Data reengineering and data transformation.
- Reengineering existing systems to produce MIS and EIS.

1.4 The Challenges

During the process of legacy system evolution, The developers inevitably face a lot of difficulties in the design and maintenance of their software systems. In general, these difficulties can be summarised as the following challenges:

- The challenge of evaluating legacy systems.
- The challenge of balancing the business needs and technical requirements of systems evolution
- The challenge of keeping the consistency of business performance.

- The challenge of developing a detailed modernisation plan.
- The challenge of understanding legacy systems.
- The challenge of discovering and extracting valuable legacy assets.
- The challenge of enabling business professionals modify integrated business knowledge.
- The challenge of involving stakeholders and reconciling their conflictions.
- The challenge of migrating legacy assets into new systems.
- The challenge of validating the success of the proposed approach.
- The challenge of supporting continuous evolution of large-scale distributed systems.

1.5 Original Contributions

Although the solutions for different legacy systems cannot be the same, but there is an underlying commonality in managing the maintenance tasks from system to system. A general consensus is that a maintenance and evolution process must be manageable and adaptive, the system architecture should be recovered and recreated using defined techniques, the change impact should be controlled and tested, and finally, the business goal should be preserved. Based on this understanding, the proposed approach introduces a methodological framework which offers a body of pragmatic business rules approach to assist legacy systems evolution. In this thesis, enterprice information system has been divided into four layers from concrete legacy system codes to the most abstract business strategy. The methodological framework overcomes the drawbacks in the existing evolution strategies by combining bottom-up legacy system reengineering approaches with top-down business comprehension methods. Thus, it can obtain a meet-in-the-middle effect to reduce the workload of software engineers.

The work advances knowledge and practice on business rules for adaptive systems. By making business requirements executable as rules, a shortcut development process

that goes directly from requirements to deployed system is demonstrated as possible. Maintenance becomes the process where business experts change text-based rules (low risk) as opposed to code change by programmers (high risk).

The desired target systems can manage and execute business rules as an independent process called upon by business rules management component. Business rules can be tested off-line and then introduced into a running system according to business strategy (scheduled, automatic or manual updates).

Most previous works on systems evolution concentrate either on technical aspects or the management issues of the topic. This thesis tries to address both issues in a balanced way. While other attempts to model business rules by considering only their acquisition, the proposed methodology supports a systematic approach to the acquisition, deployment and evolution of business rules.

For any information system developers, this proposed methodology will improve their competitive edge, and save both money and time for information system development. Indirectly, the business sectors served by the developed information systems will benefit from more timely system delivery and more efficient systems, which will bring increase in their business as well. The above impacts are predicted to happen at rather significant scale.

The contributions of this thesis are in the synthesis of the elements of the methodology, and in the automated procedure that uses the information captured in the framework in order to infer business rules. The components of the framework although individually are not new, they are being newly applied to business rules. As such, this thesis shows the application of a decision support model to the analysis and generation of business rules, and to the refinement of goal-oriented rules into operational rules, the use of inductive learning algorithms for the generation of business rules, the integration of an enterprise model with a decision support model and inductive learning techniques, and a methodology that helps resolve key issues at various stages of the business rules lifecycle.

All in all, the contributions of this thesis are concluded as the following points:

1. Introducing a pragmatic business rules approach to assist legacy systems evolution.
2. Providing opportunity for non-technical business users to take part in the business knowledge migration.
3. Drawing a novel architecture for legacy systems from business view of point. In this architecture, any legacy system can be divided into four layers: business strategies layer, business rules layer, business activities layer and infrastructure layer.
4. Analysing legacy systems from high-level structure. By studying the functionsa nd activities of legacy systems, softwer engineers can quickly locate the reusable part and improve the effiency of legacy system evolution.
5. Implementing GAs-based clustering and model dependency analysis to trace the entire code fragments that are related with reusable legacy functions.
6. Defining business rules templates to extract business rules from legacy source code. Making them understandable for non-technical business users.
7. Using ERDs to help transferring natural language represented business rules into computer reconisable XML structures.
8. Designing business rules management subsystem for handling business rules. Thus, the extracted legacy business rules can be integrated into service-oriented architecture.

1.6 Scope of Thesis

The proposed approach presented in this thesis contains both high level specification abstraction and low level source code manipulation. It covers a wide range of multidisciplinary topics from the fields of software engineering, business process modeling and service-oriented development.

The thesis explains the business rules approach, provides a step-by-step guide for designing and developing legacy system evolution in legacy system evolution, defines

the role of business rules in SOA, and discusses how business rules technique assists the web services stack and to other standards. It also covers two important BPEL (Business Process Execution Language) servers — the CRM systems and ERP system. The thesis presents the service-oriented approach to business process definition using web services, which enables software engineers to develop loosely coupled solutions. The in-depth information and practical perspective in this thesis intend to guide readers to build business rule driven information systems that can fully support the goals of their organisation.

All in all, the topics covered in this thesis include

- How the business rules approach is changing the face of business and opening new areas of opportunity
- The direct impact of business rules on IT projects.
- Understanding legacy system structure and working processes.
- Defining and discovering business rules.
- Controlling business rule quality.
- Organising business rules using rule categories and sentence patterns of rule statements.
- Working successfully with business rule management system.
- Understanding the role of business rules and models in business process management.
- Using models to structure and manage business activities, including e-commerce.
- An in-depth look at service-oriented architecture.
- Theoretical foundations, including formal constraints and predicates.
- Implementing business rules using service-oriented technology

1.7 Thesis Structure

This thesis documents some technical and managerial aspects of systems evolution, providing insights into how a modification to a system could be initiated, managed and technically achieved. This thesis also addresses how the system evolution has considerable impact on the corporate business process, personnel management and the technological advantages.

The major part of this thesis is used to explore the rationale behind business rules, as well as the many opportunities presented by this innovative new approach, and offers experience-based best practices for expressing business rules. In addition, the thesis looks at the role of business rules within the context of a wider trend towards model-based development and offers an accessible introduction to the underlying theory of business rules and notation.

This thesis is categorised into the following chapters:

Chapter 1 introduces what business-driven approaches are about. From a business perspective, the business rule approach fits. It summarises the concepts, the approach to be taken and the benefits to be gained in these two crucial technological areas.

Chapter 2 provides a detailed introduction to business rules approach and Service-Oriented Architecture (SOA). Also, new approach on the service-oriented software reengineering is addressed, which were based on business rules manipulation and by means of abstract level specification. Chapter 2 also reviews the previous works on software reengineering, whereby emphasis is placed on the progress made in the theoretical understanding of the system architecture and business rules abstract and organisation. It discusses business processes and their automation, explains the role of BPEL, web services, and Enterprise Service Buses (ESB) in SOA, provides insight into business process composition with BPEL, explains the most important features, compares BPEL to other specifications, provides an overview of BPEL servers, and discusses the future of BPEL.

Chapters 3 outlines the methodological framework. It roughly presents the theory

and practical details and illuminates the main workings of methodological framework.

Chapter 4 surveys existing legacy evolution strategies and technologies and then discusses legacy system understanding and architecture recovery. This chapter focuses on system-level partitioning method towards domain-specific business knowledge extraction.

Chapters 5 concentrates on business rules extraction.

Chapters 6 introduces how to present business rules in XML.

Chapters 7 introduces Service-Oriented BRM subsystem.

Chapters 8 discusses service-oriented platform migration and explains how to integrate business processes with BPEL.

Chapter 9 gives the details of practical implementation of proposed approach with two case studies. An extensive argumentation will also be represented in this chapter to prove the proposed methodological framework is adaptable in various experiments and realistic projects.

Finally, Chapter 10 summarises the main findings, remains open issues regarding the methodological framework, and outlines a research agenda for the future.

Chapter 2
Background and Related Research

Objectives

- To provide the background information of this thesis.
- To review the work related to business rule approach.
- To present the state of the art of software reengineering techniques.

This thesis aims to provide a business rule based approach for legacy system evolution. With the development of both academic and practical research in software reengineering, several reengineering techniques are able to be applied in the proposed approach. In this chapter, the existing techniques that related to software evolution, reengineering, business rule approach are reviewed. Software evolution of legacy system is the activity that satisfies new business requirements in an efficient method. Software Reengineering is the main technology to achieve software evolution. Business rules are the most valuable assets in legacy systems.

2.1 Service-Oriented Architecture

A Service-Oriented Architecture (SOA) is a distributed system architecture consisting of a collection of services. A service encapsulates a reusable business function, loosely-coupled to other services, and is invoked through connection technologies. As SOA can be developed incrementally (i.e., one service at a time), the fat client can be converted one screen at a time. As IT of unit of a company is under a flat headcount constraint, an SOA approach, being incremental and low-risk, was the

only way to get upper management approval.

2.1.1 Layers of SOA

SOA can be expressed very simply: SOA is about connecting customer requirements with enterprise capabilities, regardless of technology landscape or arbitrary organisational boundaries.

SOA is composed of four layers including operational resources, enterprise components, services and business processes. Following shows the description of these four layers [35].

- Operational resources: Comprised of existing systems, applications, and databases, the operational resources layer represents the legacy enterprise. Customer relationship management (CRM), enterprise resource planning (ERP), and product life-cycle management (PLM) systems are good examples of operational resources. The services that are made available through an SOA leverage these existing investments and uncover new opportunities for utilising these assets within a larger enterprise context.

- Enterprise components: Enterprise components typically employ container-based technologies such as CORBA, EJB, COM, DCOM, .NET, and so on. These assets are responsible for managing custom business logic and interfacing with the operational resource layer to carry out this logic. Additionally, they support the scalability and quality-of-service requirements of the services exposed in the layer above.

- Services: Capabilities from the enterprise component layer are selectively identified as services. The analysis, design, and development of these services is then funded and the services are deployed in order to expose these capabilities through well-defined interfaces. Service descriptions, quality-of-service (QoS), and other key service metadata are also defined to accompany these important SOA assets.

- Business processes: Individual services provide incremental value for an organisation but will likely never transform the way business gets done. Business processes, however, represent powerful orchestrations of one or more services that solve a business problem. Services are bundled together into a logical flow (described as orchestration or choreography) to solve some sort of end-to-end business problem.

2.1.2 Benefits and Challenges of SOA

Gartner advises Service-Oriented Development of Applications (SODA) is estimated to reduce total IT expenses over the long term by as much as 20% compared to traditional development methods and the savings becomes exponential over time as library of business services expands and greater degree of reuse is achieved. The primary benefits of SOA can be summarised as technology perspective and business perspective. These two types of advantages are listed in Figure 2-1.

Technology Perspective	Business Perspective
More flexible architecture	More effective integration with business partners
Integration of existing applications	Support customer service initiatives Enable employee self service
Improved data integration	Streamline the supply chain
Supports business process management	More effective use of external service providers
Facilitates enterprise portal initiatives	Facilitate global sourcing
Speeds custom application development	

Figure 2-1. Benefits of SOA

Although SOA provides lots of benefits, it SOA still faces many challenges. Four main challenges of SOA [26] are represented as follows:

- Reuse: Many of the benefits that organisations hope to achieve with SOA require that services be reused within their environment.

- Efficiency in Development: Making development more efficient means building more functionality, in less time, at less cost. Doing so depends on a variety of factors, including the reuse of services and the ability to quickly

compose applications from those services.

- Integration of Applications and Data: SOA, based on Web Services, promises to simplify integration by providing universal connectivity to existing systems and data. But, as with everything else, technology is only a small part of the solution. So the integration of existing applications and data is perhaps the most perplexing challenge facing SOA and enterprise IT organisations.

- Agility, Flexibility, and Alignment: Agility and flexibility occur when new processes can quickly and efficiently be created from the existing set of services. Achieving agility and flexibility requires an easily searchable catalog that lists the functions and data provided by the available services. In addition, an efficient way to assemble the business processes from the services needs to be available.

2.1.3 SOA and Web Service

The earlier SOAs [127] used technologies such as Distributed Component Object Model (DCOM) or Common Object Request Broker Architecture (CORBA) as connection technology. While currently Web services represent the most likely connection technology for services. There is no requirement that SOA must be implemented using Web services. In fact, a Web service does not necessarily mean that it is SOA, particularly if it is not designed to support a business oriented logical view.

With the advent of Web Services and Service-Oriented Architecture (SOA), realising enterprise integration, accelerating enterprise responsiveness to customers, automating inter-enterprise interactions and optimising the business processes of the whole supply chain become feasible. The service-oriented paradigm and Web Services technologies are rapidly emerging as the most practical approaches for integrating a wide array of manufacturing resources in the manufacturing grid environment, and efforts in the Semantic Web standards and technologies present an opportunity for automating the integration process.

In brief, the link between Web services and SOA are described as follows [83]:

- Web services provide an open standard and machine-readable model (WSDL) for creating explicit, implementation-independent descriptions of service interfaces.

- Web services provide communication mechanisms that are location-transparent and interoperable.

- Web services are evolving through BPEL4WS, document-style SOAP, and WSDL, and emerging technologies such as WS-ResourceFramework to support the technical implementation of well-designed services that encapsulate and model reusable function in a flexible manner.

2.2 Enterprise Information Systems

Today, companies rely on thousands of different software applications each with their own role to play in running a business. These different software applications run on a wide range of different platforms and operating systems, and they are implemented in different programming languages. As a result, it is very difficult for different applications to communicate with one another and share their resources in a coordinated way. Today businesses need to constantly adapt and reconfigure their IT assets, systems, and business operations to meet changing customer demands, shrink cycle times, and differentiate from competition. Enterprises need to deliver accurate and relevant information where and when it is needed and enable more predictable, higher quality business operations.

However, most enterprises have invested in packaged, legacy, and custom applications that deliver value by performing specific business functions. Unfortunately, these systems usually rely on an extremely complex and mostly ad hoc architecture consisting of monolithic silos, point-to-point connections and hard-coded interactions between systems. This makes it difficult to quickly assemble and reassemble the functions they perform as part of business processes that support new and changing business requirements. This also makes it challenging to ensure that information about events and changing conditions is available to the multiple systems that may need it in time for businesses to do something valuable with the information. Those enterprise

applications that were not designed to interoperate need to be integrated enterprise and community wide.

2.2.1 Legacy Information System

The increasing cost of managing Legacy Information System (LIS) together with the need to preserve business knowledge has meant that renovating LIS has become an important research topic over the years.

A legacy information system represents a massive, long-term business investment. Unfortunately, such systems are often brittle, slow, and non-extensible. Capturing legacy system data in a way that can support organisations into the future is an important but relatively new research area.

LISs are typically the backbone of the information flow of an organisation and the main vehicle for consolidating business information. They are thus mission critical, and their failure can have a serious impact on business. In fact, a LIS can be defined as “any information system that significantly resists modification and evolution”.

- LISs usually run on obsolete hardware that is slow and expensive to maintain.
- Software maintenance can also be expensive, because documentation and understanding of system details is often lacking and tracing faults is costly and time consuming.
- A lack of clean interfaces makes integrating LISs with other systems difficult.
- LISs are also difficult, if not impossible, to extend.

Several solutions have been proposed to these problems. These solutions fall generally into three categories: *redevelopment*, which rewrites existing applications; *wrapping*, which provides a new interface to a component, making it more easily accessible by other software components; and *migration*, which moves the LIS to a more flexible environment, while retaining the data and functionality of theoriginal system.

Given the scale, complexity, and risk of failure in LIS projects, a well defined, easily implemented, and detailed methodology is essential to project success. However, few comprehensive LIS migration methodologies are available, and a general approach has yet to be agreed on. Existing approaches are either too high level or have yet to be applied in practice [21, 49, 58]. Although partial solutions such as wrapping are widely adopted, such solutions are short term and can actually complicate LIS maintenance and management over the long term. On the other hand, redevelopment approaches tend to be too risky for most organisations. Companies thus advocate migration as a sound strategy for contending with LISs. Here an overview of existing LIS strategies is offer, then migration issues are discuss and two promising approaches for migrating LIS data are present.

Constant restructuring and alteration is often carried out to keep them functionally operational, however lack of system knowledge and appropriate documentation has meant that certain systems continue to run on their original specification. While such systems are often viewed as prime candidates for replacement; having been in operation for a number of years, these systems have also become mission-critical to the organisation. Not only do they perform important functions for the organisation but in some situations they are the only source of business knowledge (for example in the form of business rules, work processes and more importantly business data).

The major issue with current approaches is that they view the LIS problem primarily from a technical perspective. The limitation with such a perspective is that it not only ignores the organisational and business aspects of LIS, but it also means that the whole effort is spent on creating a LIS for the future. The business knowledge, often vital for the organisation, is either neglected or always tied into some technology. As a result, while on one hand, the program understanding task becomes that much more difficult and costly to implement, on the other hand, it also means that little or no effort is spent on preserving or sophisticating (improving) the business knowledge in a manner that allows for its repeated use.

Because a LIS already meets some of the business and user requirements demanded of the target system, it is important to understand its operations and interactions.

Additionally, poor LIS understanding can lead to incorrect target-system requirement specifications and ultimately to failed migration projects. Hence it is necessary to capture and document such business knowledge into business models, where the knowledge can be understood and reflected through a business perspective and allowed to evolve in line with business needs.

Furthermore, to obtain business knowledge for legacy information systems has two distinct advantages. Firstly, it can deliver a system whose business knowledge is probably more sophisticated than that of the pre-existing system. Secondly, since the approach is model-driven, the achieved benefits outlive the individual project and can be reemployed in subsequent renovation projects.

Thus in order to have a good understanding of the LIS data, interfaces and applications, tool support is requied [34, 11]. However, although some support is recently available, engineers may have to develop specialised tools to fit their LIS and target systems [10].

2.2.2 Enterprise Application Integration

Business objects, as mentioned earlier, refer to the objects that exist in the real world and as a result have meaning and usage to the business.

As enterprise departments interact on a real-time basis and business processes cut across multiple enterprises and business lines, the need for information integration leads to the adoption of Enterprise Application Integration (EAI) [65] approach. EAI is the process of coordinating the operation of various applications across an enterprise. EAI is the process of creating an integrated infrastructure for linking disparate systems, applications, and data sources across companies. EAI entails integrating applications and enterprise data sources so that they can easily share business processes and data. Integrating the applications and data sources must be accomplished without requiring significant changes to these existing applications and data. Thus, legacy and newer systems can integrate to provide the business with greater competitive advantage.

Originally EAI [19] was only focused around integrating Enterprise Resources

Planning systems with other applications within enterprise, but now the volatile nature of business requires the approach can be used as a catch-all term to cover all the other aspects of business integration. Therefore, the EAI-based integration has shown its short come in solving the interoperability issues. The requirements of accessing underlying data and business logic lead to EAI evolution. The solution should comprise the challenge of efficiently integrating diverse business processes and data across the enterprises, allowing the organisations to keep pace with and respond to market changes. The evolution of EAI approaches aims at service-oriented functionality of enterprise applications which enabling technologies ranging from EDI to Web Services. XML based process integration are analysed to provide flexible, scalable and adaptable EAI framework.

EAI has been pursued strenuously by researchers and industrial practitioners for decades. The most renowned network computing, middleware and relevant technologies used for EAI could not provide a complete solution to holistic enterprise integration, but have contributed more or less to the development and emergence of more promising technologies.

In the current business environment, vital information exists in many disparate systems throughout the enterprise. These include:

- Customer Relationship Management (CRM) systems
- Financial and accounting systems
- Enterprise Resource Planning (ERP) systems
- Web servers
- Legacy systems

This information must be accessible from multiple touch-points. These systems typically have evolved internally in isolated silos. Despite their focus on product lines and departmental functions rather than customer needs, these systems nonetheless contain strategic information.

Until now, if this company wanted to integrate these different systems, it had to employ developers to create custom bridging software to allow the different applications to communicate with one another. However, these sorts of solutions are often piecemeal and time consuming. As soon as a change is made to one application, corresponding changes have to be made to the other applications linked to it and to the bridges that link the applications together.

2.2.3 Evolution of Enterprise Application Integration

The approaches in this camp only consider the basic process of migrating the data from one system to another: Where effort is undertaken to understanding the semantics, the focus is necessarily limited to making improvements from an application perspective (schemas and data normalisation aspects) and not from a business perspective (real world reference). Consequently, issues such as integration and harmonisation of application data from various LIS remain largely unresolved primarily due to lack of clear semantics and standardised data.

Current solutions rarely make use of models and architectures which allow for specifications to be laid out clearly and understood from a business perspective and used for future reference. As a result, there is hardly any attempt to extract the business knowledge and document it in business models so as to explicitly capture and separate the business knowledge from the underlying technology and enrich the knowledge.

Such a view is important as modern organisations are in a constant state of evolution and business requirements are continuously changing. In order to understand and represent business requirements, it is important to capture the relative knowledge in models that allow an organisation to understand the inherent composition of the business, along with a clearly defined way to deliver business value, irrespective of particular application concerns.

2.2.4 Enterprise Business Process Management Solution

Business Process Management (BPM) [73] software arranges the services that are

exposed in SOA in a particular order and guides them according to certain rules and conditions. In addition, BPM provides graphical tools for mapping these process model flows. In some organisations, business users, rather than the technical staff, are already using these tools to model, simulate, and manage the process-centric business applications of organisation. However, while SOA provides a large set of services for BPM to tie together, it is rare for all the nodes in a process diagram to be based entirely on existing enterprise services. This is why BPM technology must also seamlessly mix these SOA-based services with human tasks and calls to other service technologies, such as messaging systems.

Enterprise BPM suites facilitate inclusion of SOA-based services into process diagrams by shipping with Web service integration nodes. These nodes let a process designer search public or private Universal Description, Discovery, and Integration (UDDI) directories for a desired Web service. It then provides interfaces for mapping the input and output parameters into variables stored in the process. Additionally, enterprise BPM suites expose their own logic as consumable services by publishing execution calls to new process models and rules to a private, local UDDI. This means that as new processes are defined, the UDDI is updated automatically so users and developers with appropriate permissions can find and kick off these processes in their applications, using Web services.

2.3 Software Evolution

Twenty years ago, software needed to be corrected occasionally and a new release issued perhaps once a year. Developers could use the term maintenance to imply to that they were working to enable software to continue to do what it used to do. Ten years ago, software needed a major release with new functionality twice a year, and developers used the term reengineering to imply that they were adding new user-required functions to the software. Today, software needs to be changed on an ongoing basis with major enhancements required on a short timescale in order to meet new business opportunities and reduce the ‘‘time to market’’ for new products and services. Thus, evolution better describes a situation in which maintenance and reengineering are needed frequently [30].

In brief, Software evolution is a long-term process involving the execution, usage, improvement, extension and update of software systems [8]. It is the process of conducting continuous software reengineering. Reengineering implies a single change cycle, but evolution can go on forever. In other words, to a large extent, software evolution is repeated software reengineering.

2.3.1 Legacy System

Legacy systems are systems that are still valuable, but are notoriously difficult to maintain [42]. It is necessary to comprehend legacy systems, refine the users' requirements, and model them in order to maintain, modify and reuse them [39].

There are many different descriptions towards legacy systems. Legacy systems are large software systems that people do not know how to deal with, but that are vital to the organisation [15]. A legacy system could be any information system that significantly resists modification and evolution to meet new and constantly changing business requirements [27]. It is critical software that cannot be modified efficiently. It is a system that was developed at some time in the past and which is critical to the business in which the system operates [109]. Many legacy systems remain supportive of core business functions and are "indispensable" to the business [90].

No matter what definition legacy system is given, legacy systems are believed to be not useless. On the contrary, they are with great value and quite essential.

Legacy systems have static and dynamic aspects. The static aspects include software elements and their relationships. The dynamic aspects mainly concern the sequential events that perform tasks. In static modelling, high-level elements are found and subsystems or other logically connected software elements may be represented. In dynamic modelling, behavioural descriptions that show interactions amongst high-level static elements are abstracted.

Some people may think legacy systems are systems implemented with COBOL, however, hundreds of legacy systems are written in other languages, such as assembler and C. Along with the development of technology legacy systems have been expanded

to contain Java, XML, C++, Fortran and other coding discipline. Thus, a legacy system should be defined with the following characteristics:

- Large and complex systems;
- Geriatric;
- Written in COBOL PL/1 or any other modern language;
- Use of legacy database services;
- Hard to maintain and evolve;
- The documentation is often missing or unreliable;
- Mission critical for organisation;
- Must be operating at all times.

Besides, legacy systems are believed to progress through a sequence of life stages from their introduction through growth, maturity, and decline. The typical sequence of stages for legacy systems, which is called the enterprise system lifecycle, is illustrated in Figure 2-2.

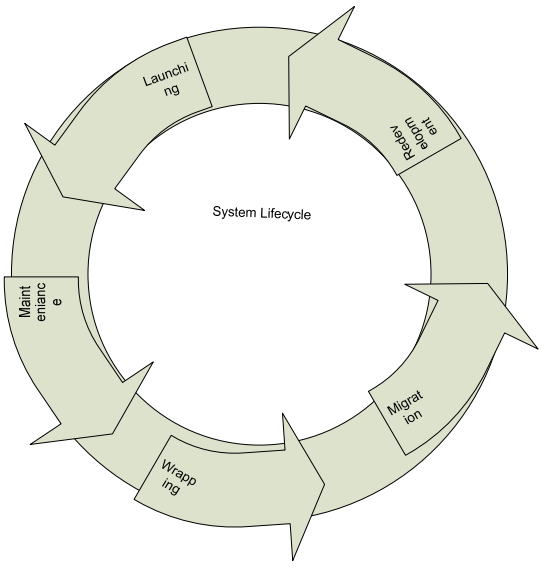

Figure 2-2. Enterprise System Lifecycle

2.3.2 Software Maintenance

Any computing system, including hardware and software systems, will inevitably grow in scale and functionality. Because of this complexity, the likelihood of subtle errors is much greater. Moreover, some of these errors may cause catastrophic loss of money, time, or even human life. Thus, large systems are so complex that it is impossible for a single individual to build and maintain all aspects of the design of system.

A major goal of software maintenance is to enable developers to construct systems that operate reliably despite this complexity. One way of achieving this goal is using formal methods, mathematically-based languages, techniques, and tools for specifying and verifying software systems. Using formal methods does not guarantee correctness, but they can greatly increase the understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected.

Most captured design information is informal (not machine-interpretable). While this is valuable for the software maintainers, informal information is subject to wide interpretation, and so the usability is limited by the problem of different software developers using different interpretations of the same description. Formal descriptions of the design (and its rationale) with a precise semantics can overcome this communication problem. It can even allow engineers to modify the design rather than the code and, thus, to modify a software system using semiautomatic tools. Transformational development of software seems to be the right way to get such descriptions of a design.

In such an ideal setting there would be no reason for reverse engineering. However, the large amount of existing software that has to be maintained forces us to face problems. Mature software systems often have incomplete, incorrect or even nonexistent design documentation. This makes it difficult to understand what the system is doing, why it is doing it, how the work is performed, and why it is coded that way. Consequently mature systems are hard to modify and the modifications are difficult to validate.

Following shows the categories of software maintenance tools:

- Configuration management
- Monitoring/evaluation
- Redesign
- Code production/analysis
- Verification/validation
- Testing/integration
- Documentation

Furthermore, there are several approaches to achieve software maintenance, including redevelopment, wrapping and migration.

- **Redevelopment**

Redevelopment, which is commonly referred to as Big Bang and also known as Cold Turkey [72], redevelops LIS from scratch using a new hardware platform and modern architecture, tools, and databases. As an example, the Renaissance research project is aimed at developing a systematic method for system evolution and reengineering [48]. The project defines a set of activities and tasks to support an overall reengineering project and the control flow between the identified activities, which drives the cooperation among tasks. Renaissance identifies generic activities that can be specialised for both the organisation and the system it is implementing.

Scott Tilley proposes a high-level framework for LIS reengineering from several perspectives [39]. For each perspective, Tilley divides the reengineering problem into phases, listing pertinent issues for each one. Although this approach does offer some guidance, it is far too high level to be practically applied.

Narsim Ganti and William Brayman [75] propose general guidelines for transforming from a centralised legacy environment to a distributed environment. In this approach, select business processes are reengineered as required and then linked to LISs,

which provide valued data and business logic to the new environment. New applications are then developed to fit these processes. This approach recognises that LIS migration should cause as little disruption as possible in the current business environment. However, it is not clear how cut-over to a separately developed target system would be handled.

In reality, the risk of failure is usually too great for organisations to seriously contemplate a redevelopment approach. Another very real concern stems from the fact that technology and business requirements are constantly changing. Thus, at the end of a long process, an organisation might find itself with a redeveloped system based on obsolete technology that no longer meets its business needs.

- **Wrapping**

Given the drawbacks of redevelopment, many organisations are forced to seek alternative ways to cope with their LISs. Most practical solutions focus on wrapping, which surrounds existing data, individual programs, application systems, and interfaces with new interfaces. In essence, this gives old components new operations or a "new and improved" look [16]. The wrapped component acts as a server, performing some function required by an external client that does not need to know how the service is implemented [8]. Wrapping lets organisations reuse well-tested components that they trust and leverage their massive investments in the LIS.

The most widely used implementation of wrapping is screen scraping, which replaces character-based front end with a client-based graphical user interface of a LIS [121]. Implementing the GUI cheaply and effectively leverages legacy data and lets users employ common graphical data manipulation tools to input data and process system output.

It is still very much a short-term solution despite the commercial success of the screen scraping. Implementing a GUI in a LIS does not address many of the serious problems such systems face, including overloading, static functionality, and high maintenance costs. In many cases, screen scraping actually compounds a maintenance problems of organisation, as the functionally superfluous screen-scraping software will

itself require maintenance.

- **Migration**

When redevelopment is unacceptably risky and wrapping is unsuitable, migrating the LIS to an open environment can be the best alternative. Although it is a much more complex undertaking than wrapping, if successful, the long-term benefits of migration are also greater. For example, migration offers more flexibility, better system understanding, easier maintenance, and reduced costs.

Although migrating LISs is a major research and business issue, there are few comprehensive approaches to migration. Given the bewildering array of LISs in operation and the problems they pose, it seems unlikely that a single generic migration method would be suitable for all systems. However, a set of comprehensive guidelines to drive migration is essential.

Before embarking on a migration project, engineers, management, and users should undertake an intensive study to find the most appropriate approach for solving LIS problems of their organisation. To the best of knowledge, the literature contains no successful, practical experience reports from projects using a comprehensive migration approach. The few successful migration-like projects reported in the literature describe ad hoc solutions to the problem.

2.3.3 Software Evolution and Reengineering

Legacy systems are valuable assets for organisations. To maintain the value of these assets, legacy systems must be changed and updated. In an organisation, the majority of the budget is devoted to evolving existing software rather than developing new software. So it is necessary to study the technologies which used to evolve software.

Reengineering is the most important technology to accomplish software evolution. Reengineering can be comprehended as three stages comprising reverse engineer; making changes to specification and design to complete the logical model; generating

new systems from revised specification and design.

As a combination of reverse engineering and forward engineering, software reengineering technology is a practical solution for the problem of evolving existing computing systems. Dynamic change management of software systems has been largely performed by using ad hoc techniques that are normally rather expensive and in some cases, impossible (if the designer has not documented the system and has left the company).

The primary activities of reengineering are listed as follows:

- Source code translation: Convert code to a new language;
- Reverse engineering: Analyse the program to understand it;
- Program structure improvement: Restructure automatically for understandability;
- Program modularisation: Reorganise the program structure;
- Data reengineering: Clean-up and restructure system data.

Reengineering is also known as restructuring or rewriting part or all of a legacy system plus changing its functionality according to new requirements. Contrary to developing new systems, reengineering has two main advantages during the process of software maintenance and evolution.

- Reduced risk

There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

- Reduced cost

The cost of re-engineering is often significantly less than the costs of developing new software.

In conclusion, software evolution is the process of continuous software

reengineering, and also software evolution can be comprehended as repeated software reengineering.

2.4 Software Reengineering

Over the past decades, there has been a tremendous investment made in information systems. Such systems have evolved from file systems, through database systems, and now emerging the enterprise information systems. With the advent of each new form of technology, there has been a need to redesign and re-implement the existing information systems.

In recent years, a great many resources have been put into the area of reengineering. Reengineering is the process of analysing existing application programs and database code to create higher-level representations of the code. Reengineering involves the redesign of existing information systems, while using as much of the existing systems as possible. That is, the approach taken is to transform the existing information systems into the format needed for the new technology, rather than to throw away the old systems. Such an approach has obvious benefits, particularly if it can be automated supported by methods and tools.

Many development teams in IT industry have moved to a revolutionary software development process called software reengineering. As a combination of both reverse engineering and forward engineering, software reengineering technology is just a panacea for the above evolution of existing computing systems.

2.4.1 Software Reengineering Theory

Software reengineering is defined by Arnold [13] as the activity that improves the understanding of software, and examination and alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form, usually for increased maintainability, reusability, or evolvability. From the definition, the software reengineering technique can be looked as goal oriented rather than technology oriented.

Software reengineering is code understanding and improvement to enable code evolution. Software reengineering seeks to clarify understanding of software, alter characteristics of code, or change the functionality of software. Reengineering is like maintenance in the aspect of it operating on existing software. Both reengineering and maintenance are unlike software development. Development creates new software requirements, design, and implementation from whole architecture, while reengineering and maintenance mend and alter an existing system. Because software reengineering is concerned with software already in existence, the reengineering techniques can operate only on software characteristics which are actually found in software. This differs from new development, where much of the information and process used to successfully create the software does not appear as part of the end software product.

A major goal of software reengineering is to enable developers to construct systems that operate reliably despite the complexity. One way of achieving this goal is by using formal methods, which are mathematics based languages, techniques, and tools for specifying and verifying such systems. Use of formal methods does not guarantee correctness. However, they can greatly increase the understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected.

There are at least two advantages of using formal methods as the foundation of software reengineering. First, formal methods can help software engineers to acquire a rigorous and precise description of the system being reengineered, therefore greatly increasing the quality of the new system. Second, automation is one of the key goals of reengineering. By applying formal methods, it may be possible to automate more of the process of reengineering.

These two advantages will naturally bring flexibility and reliability to the reengineered software, because future reengineering can still enjoy automated tool support, and reliability is enhanced by the rigor and precision provided by formal methods.

Since all reengineering techniques assume that the only information available is from the software itself, reengineering abstraction levels and primitive operations need

not be as comprehensive as the levels and operations used in new development. The limited access to information of reengineering plays a central role in limiting the scope of the following discussions. Software reengineering techniques fitted for a variety of purposes, from code re-documentation to language translation.

Following shows the classifications of Reengineering tool:

- Browsing tools (search)
- Code improvement tools (reformatting and restructuring)
- Reverse engineering (abstraction)

In all, software reengineering means implementing or re-implementing a part of a legacy application using modern technologies, without modifying overall functionality. It is a systematic way of examination and alteration of a subject system to reconstitute it in a new form and the subsequent implement of the new form [101]. The goal of software reengineering is to restructrue or rewrite legacy systems into new systems with better maintainability, flexibility and scalability [62]. Many technologies are proposed to support software reengineering, such as forward engineering and program transformation.

2.4.2 Software Abstraction Level Definitions

Axioms, requirements, design, and implementation are the levels of software abstraction. Figure 2-3 illustrates that the broad scope of axioms are distilled through requirements to the many levels of design (represented by broken lines) and down to implementation. The information in “lower” abstraction levels is more precise and refined than at “higher” levels. Details are predominant at lower levels; possibilities abound at higher levels. Examples of possible corresponding languages for each level are stated on the side.

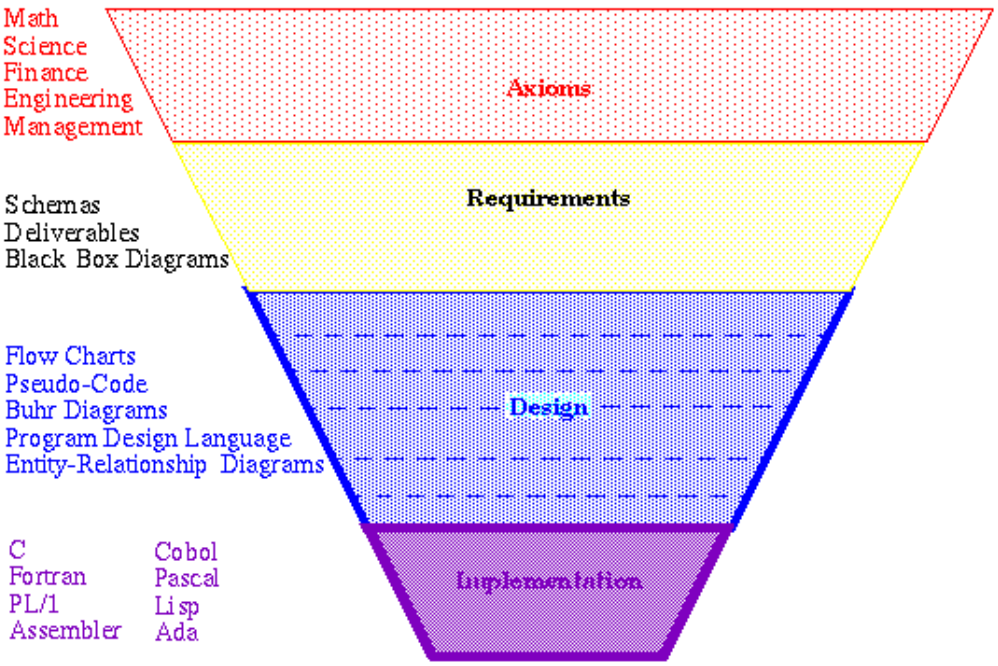

Figure 2-3. Levels of Software Abstraction

- Axioms

An axiom is any fundamental, self-consistent, universally accepted tenet about the context of software. Axioms may include the purpose of software, the concepts that imply the requirements (but not the requirements themselves), and details on the source and nature of information to be processed by the software. Axioms place software against a background of non-software subjects. They explain the nature and purpose of the software without providing details about the actual software, as in requirements, design, and implementation. A complete software axiom set allows the creation of a complete set of requirements through appropriate forward engineering operations without additional information.

- Requirements

A requirement is a well-formed statement with respect to the axioms of the conditions necessary for successful software use, conditions prevailing after software use, and constraints on software use. Requirements state what the software does with the resources provided, not how the software does it. A complete software requirement set allows the creation of a complete design through appropriate forward engineering

operations without additional information.

- Design

A software design is any non-compliable software description that faithfully reflects requirements and states how outputs are produced from inputs. Generally, many design levels sequentially provide greater and greater detail about how software operates. Each successive design level refines requirements into implementations. The design defines software architectures, organisation, communication, and other details about how software acts without containing enough detail to actually be compliable. A complete software design allows the creation of a complete implementation through appropriate forward engineering operations.

- Implementation

A software implementation faithfully reflects design and is syntactically correct with respect to a compliable language(s). A complete implementation could be compiled to produce an executable code. An implementation must be syntactically correct in a compliable language(s), but may be fragmentary. That is, the source code could consist of a set of nested “if-then-else” that are syntactically correct but are not compliable without additional information such as variable declarations, file headings, and other code elements. Such an implementation, with just the “if-then-else”, is syntactically correct but is only a fragment of a compliable set of source code. A complete implementation must actually be compliable, and therefore contain all elements needed for compilation. A complete implementation needs more than just syntactically correct code fragments.

2.4.3 Program Transformation

A transformation operation is any correctness preserving operation on S that maps a subset of one implementation into another implementation, one design into another design, or one set of requirements into another set of requirements.

Over time the program code is modified to remove errors and to adapt the system to

changed requirements. The requirements may change to allow usage of alphanumeric keys and to be able to handle large amounts of data, and the implementation revised to use disk-based B-trees. Unfortunately, often these changes take place without being reflected correctly in the specification. The gap between the original specification and the program becomes larger and larger. The result is a program code without a proper specification and with untrustworthy design information. The code becomes difficult to understand and, thus, difficult to maintain. To overcome this deficiency, it is important to change the specification first and then reflect the changes in the program code. Thus program transformation, software maintenance and evolution by changing a specification and re-synthesising, is proper to be applied to achieve the change between specifications or source code in other layers.

A program transformation [55] is any operation that takes a program and generates another program. It is often important that the derived program be semantically equivalent to the original, relative to a particular formal semantics. Other program transformations may generate programs that semantically differ from the original in predictable ways. The languages in which the program is transformed and the resulting program are called the source and target language respectively [108].

Program transformation can be used in various areas of software engineering including refactoring, program synthesis, software renovation, reverse engineering, optimisation and so on. The taxonomy of program transformation is listed in Figure 2-4.

Translation	**Rephrasing**
Migration	Normalisation
Synthesis	*Simplification*
Refinement	*Desugaring*
Compilation	*Weaving*
Reverse engineering	Optimisation
Abstraction	*Specialisation*
Decompilation	*Inlining*
Architecture extraction	*Fusion*
Software visualization	Refactoring
Analysis	*Design improvement*
Control-flow analysis	*Obfuscation*
Data-flow analysis	Renovation

Figure 2-4. Taxonomy of Program Transformation Straction

2.4.4 Program Slicing

In 1979, program slicing [119] was defined by Weiser to be a compute slice by computing consecutive sets of indirectly relevant statement according to data flow and control flow dependences [27]. Moreover, a program slice can also be found in linear time as the transitive closure of a dependency graph.

Program slicing is a family of program decomposition techniques based on selecting statements relevant to a computation, even if they are scattered throughout the program.

There are two kinds of basic slicing including static slicing and dynamic slicing [40]. Unfortunately, static slicing does not allow determining which particular execution paths in the legacy code yield correct and which ones anomalous results. Consequently, code slices may actually contain statements that are not relevant for successful completion of a program thread. Dynamic slicing overcomes this predicament by cautiously examining only those variables that are relevant in a specific program thread. Program slicing is a relatively well-understood technique.

Except for the two types of basic slicing, there are several other kinds of slicing such as conditioned slicing [73, 80], amorphous slicing [66, 155].

Conventional program slicing has been also advocated for the purpose of software reuse. However, program slices are often imprecise as reusable functions because they contain unnecessary statements for the function to be recovered. Hence this proposal of a new slicing approach, called transform slicing, which is more effective in extracting functional components from old programs [131].

Debugging is the original motivation for program slicing, however, program slicing has been applied in more other problem areas consisting of cohesion measurement [17], software comprehension [59], software maintenance and reengineering [87, 129].

That is, program slicing has been applied in program debugging, parallel processing, program testing, program integration, program understanding and software maintenance,

both using the basic definition and developing variants, including program dicing, dynamic slicing, decomposition slicing, relevant slicing , interface slicing , conditioned slicing , and variable slicing.

For instance, Lakhotia and Deprez [111] introduced a program tucking transformation, which uses slicing and other code isolation techniques to extract code components with the aim of improving the cohesion and coupling of the program. A software “wedge” is driven into the code to bind the area of interest. Statements which concern a particular computation are sliced out of a procedure and folded into a new procedure. The sequence of transformations wedge-slice-fold constructs a “tuck” transformation.

2.4.5 Clustering

Clustering [117] is a well-known technique for grouping possible combinations of data attributes and methods into classes [33]. It is concerned with grouping similar entities into constellations (clusters) with strong links among its members and weak links with entities belonging to other constellations. Similarity of entities in clusters can be determined in two ways [24]. First, the relationships between entities may serve as an indication of their similarity. The closer the entities are with respect to some reference point, for instance, a certain point in space, the more similar they are. Second, a similarity coefficient may be computed for the entities based on multivariate analysis.

There are a lot of clustering method have been proposed in the reverse engineering literature, each of which uses a different induction principle. Generally, they are different in the following three aspects:

- Entity Types

The granularity level of the selected resources depends on the purpose of the cluster analysis. Procedures, functions, data types and variables (lower-level of granularity) are usually used for clustering at the module level; classes, modules and components (higher-level of granularity) are used for clustering at the *subsystem level*.

- Proximity Measures

Since clustering is the grouping of similar instances/objects, Distance Measures and Similarity Measures are used to determine whether two objects are similar or dissimilar. The first method measures the distance between two entities by calculating the number of operations needed to transform one entity into the other. The second method is based on relationships and shared properties between two entities (e.g. function call, features or data use) in which the similarity is measured as a function of the number of static occurrences of such relationships [135].

- Clustering Algorithms

Important clustering algorithms fall into two general categories: Hierarchical Algorithms [95] and Optimisation Algorithms. In Hierarchical Algorithms, each entity is first placed in a separate cluster and then gradually the clusters are merged into larger and larger clusters until all entities belong to a single cluster. Optimisation Algorithms partition the whole system into clusters and improve the clusters towards an optimal partition by iteratively relocating entities among the clusters.

Considering the above aspects with the work that has finished so far, it is clear that an adequate clustering approach applied in current context should include three factors: cluster at higher level of granularity, use similarity measuring method and implement optimisation clustering algorithms. In this way, cluster analysis can be applied to regroup the recovered legacy entities into subsystems with a high internal cohesion and low coupling between subsystems [22, 143].

2.4.6 Information-Flow Analysis

Identifying domain variables automatically from legacy code is an application of information-flow relations algorithm [57]. The identifying solution is divided into three steps: identifying domain variables affected by input domain variables; identifying domain variables which affect output domain variables; domain variables management. It will help to extract business rules more effectively and correctly. The solution has been applied to a large complex financial legacy system which has proved to be

successful.

In most situations, the modules that contain domain knowledge are composed of four kinds of program statements: assignment, compound, conditional and repetitive statements. Table 2-1 shows how to derive construction rules of information-flow relations for each kind of statements.

Statement Type	Information-Flow Formula
Empty Statements	$\lambda = \varnothing; \quad \mu = \varnothing; \quad \rho = \iota$
Assignment Statements	$\lambda = \Gamma(e) \times \{e\}; \quad \mu = \{(e,v)\}; \quad \rho = \Gamma(e) \times \{v\} \bigcup (\iota - \{v,v\})$
Compound Statements	$\lambda = \lambda_A \bigcup \rho_A \lambda_B; \quad \mu = \mu_A \rho_B \bigcup \mu_B; \quad \rho = \rho_A \rho_B$
Conditional Statements	$\lambda = (\Gamma(e) \times \{e\}) \bigcup \lambda_A \bigcup \lambda_B; \quad \mu = (\{e\} \times (D_A \bigcup D_B)) \bigcup \mu_A \bigcup \mu_B$ $\rho = (\Gamma(e) \times (D_A \bigcup D_B)) \bigcup \rho_a \bigcup \rho_B$
Repetitive Statements	$\lambda = \rho_A^*((\Gamma(e) \times \{e\}) \bigcup \lambda_A); \quad \mu = (\{e\} \times D_A) \bigcup \mu_A \rho_A^*((\Gamma(e) \times D_A) \bigcup \iota);$ $\rho = \rho_A^*((\Gamma(e) \times D_A) \bigcup \iota)$

Table 2-1. Definition of Information-Flow Relations

Various construction rules are implemented to identify information-flow from assignment, compound, conditional and repetitive statements; by repeated application of these rules it is possible to obtain the flow relations for the complete statement part of a procedure or program.

The information-flow relations are presented, with their construction rules for assignment, compound, conditional, and repetitive statements; by repeated application of these rules it is possible to obtain the flow relations for the complete statement part of a procedure or program.

In Method Dependence Graphs (MDGs), each module responses to a specific task. Two types of dependence between variables-data dependence and control dependence are captured. Data dependence occurs between two variables when one is used to define

the other. Control dependence occurs between two variables when the assignment of one variable is controlled by the value of the other variable.

An efficient solution to identify domain variables is implementing Data Dependence Graph (DDG). DDG is the graph to represent data dependencies relation of variables. Thanks to the legacy system partitioning stage, reverse engineers can easily extract DDGs from MDGs. The MDGs not only represent operation dependences between modules, they also make explicit both the data and control dependences for each operation in a legacy system. The statements and expressions of a program constitute the vertices of a MDG, and edges correspond to data dependences and control dependences between statements. To get DDGs, here reverse engineers need to delete the control dependence from MDGs.

The next task is separating data dependence and control dependence. For each call statement, there is a call-site vertex in the MDG as well as actual-in and actual-out vertices which mode the copying of actual parameters to or from temporary variables. The control dependence will be described as assistant information in domain variables management.

Precondition of building DDG is to generate Control Flow Graph (CFG). A CFG [14, 15] has been the usual representation for the control flow relationships of a program. There are two ways to build DDG from CFG. One is directly generation from CFG using the information of data dependence. The sets DEF(i) and REF(i) denote the sets of variables defined and referenced at CFG node i. Node j is data dependent on node i if there exist a variable x such that:

1) $x \in DEF(i)$

2) $x \in REF(G)$ and

3) there exist a path from i to j without intervening definitions of x. So the Data Dependence Graph (DDG) can be derived for CFG

Normally, business rules are defined as the form of ECA; they can be recognised as functions, constraints or transformations to generate application outputs from inputs.

This is so because business rules must eventually produce some outputs to their intended users and they must take some inputs form their users to start the process. After the source segment has been cut out of the procedure division, a data flow analysis is performed to recognise all data variables processed by that segment.

2.5 Business Rule Approach

Software systems become legacy systems when they cannot adapt to new requirements. However, the knowledge embodied in legacy systems constitutes a significant corporate asset. Assuming that these systems still provide significant business value, they must then be modernised or replaced. Business rules approach describes a knowledge management approach to legacy system evolution that applies the knowledge of software engineering within a business context.

2.5.1 Principles of Business Rule Approach

A business rules approach is a formal way of managing and automating business rules of enterprises, so that the business behaves and evolves as its leaders intend. Also, a business rules approach to systems development promises to be the most practical and desirable way to build systems. It can help software engineers build better, easily changeable systems faster than any previous approach. Thus business rules are used to capture and implement precise business logic in processes, procedures, and systems (manual or automated), and can also provide the basis for expert systems.

According to Wikipedia, Business rules describe the operations, definitions and constraints that apply to an organisation in achieving its goals. These rules are used to help the organisation to better achieve goals, communicate among principals and agents, communicate between the organisation and interested third parties, demonstrate fulfillment of legal obligations, operate more efficiently, automate operations, perform analysis on current practices, etc. [21]. The whole point of the business rule approach is that it identifies the conditions that need to be detected and the corrective action that is required to restore the system to a state in which “true” always prevails.

Business rules arise from the objects that one encounters in a business and their interrelationships. These “business objects” may be found in documentation, procedure manuals, automation systems, business records, or even in the tacit know-how of staff. It is these objects that are modelled by the domain ontology objects.

The Basic Principles of the Business Rule Approach is listed as follows:

- Rules should be written and made explicit.
- Rules should be expressed in plain language.
- Rules should exist independent of procedures and workflows.
- Rules should build on facts, and facts should build on concepts as represented by terms.
- Rules should guide or influence behavior in desired ways.
- Rules should be motivated by identifiable and important business factors.
- Rules should be accessible to authorised parties.
- Rules should be single-sourced.
- Rules should be specified directly by those people who have relevant knowledge.
- Rules should be managed.

2.5.2 Business Rule Engines

Rule-based programming is a declarative programming technique in which developers can write heuristic rules rather than procedural instructions. A software module called a rule engine decides how to apply the rules to data. Rule-based programming is widely used in the insurance and financial services industries, and is especially useful whenever complex criteria must be applied to large amounts of information.

Business rule engine is the term used to describe software used to track, manage and revise enterprise business processes. A business rules engine is a software system that executes one or more business rules in a runtime production environment. This kind of software system typically executes business rules that might be derived from legal regulations or in-house corporate policies. Business Rule Engines (BREs) can be used alone or in conjunction with other technologies, like Business Process Management (BPM). There are many commercial rule-based engines out there such as Advisor, Haley's, Jess, and JRules as well as open source software: CLIPS, Drools, Mandarax, and so forth. These products could be used in refactoring business rules from application code.

Rule engines can be used together with various commercial off-the-shelf products, such as application servers. Historically, there has been a certain amount of vendor lock-in in choosing a rule engine, because each rule engine has its own interface of programmer.

The Java Rule Engine API, defined by the *javax.rules* package, is a standard enterprise API for accessing rule engines, currently being developed by a consortium of rule engine vendors under the Java Community Process. The *javax.rules* package will allow the host program to interact generically with multiple rule engines, the same way the Java Database Connectivity (JDBC) API makes it possible to write vendor-neutral database programs. The API will include mechanisms for creating and managing sets of rules; for adding, removing, and modifying objects in working memory; and for initialising, resetting, and running the engine.

The *javax.rules* package can be used in both J2SE and J2EE applications. Using the *javax.rules* package, Java code can be written to work with rule engines in a generic manner, so that changing from one rule engine to another will not necessitate changes to Java code. The name JSR 94 refers to the standards committee that developed this new API.

The specification for the Java Rule Engine API (JSR 94) has been developed through the Java Community Process (JCP) program. It defines a Java runtime API for rule engines by providing a simple API to access a rule engine from a Java application.

JSR 94 provides guidelines for the rule administration and rule runtime APIs, but it defines no guidelines for what language to use to define the rules and actions. Efforts are under way to standardise a common rule language, including the Rule Markup Language (RuleML).

Implementations of JSR 94 intended for use with J2EE containers deal with the programming restrictions imposed by that environment. Other JSR 94 implementations will be targeted toward high performance in J2SE applications, and some will switch between the two.

JSR-94 allows developers to interface with any rule engine that supports the standard. It is also useful from a pure code reading perspective. According to the JSR 94 specification, the rule engine adapter divides interaction with rule engines into *administrative* and *runtime* interactions.

The rule administration interface is defined in *javax.rules.admin* package and primarily used for loading and managing rule execution sets from external resources like URI, XML Element and binary abstract syntax tree.

The rule runtime interface, which defined in the *javax.rules* package, provides classes to be used by clients to execute specific rule sets and get results. It represents connection between client and specific rule engine by using a *RuleSession* object.

A packaging separation between the runtime client API and the administration API was made to reinforce the distinction between executing a rule execution set that has been previously loaded and registered into the runtime environment by an administrator, and the dynamic loading and execution of external resources. The later actions can only be performed using the classed and interface in the *javax.rules.admin* package. The distinction between the runtime and admin package allows a more fine-grained control of the user population, for example, some users may be allowed to execute rules but not to administer them.

2.5.3 Business Rule Language

Today, business rules are the focus of various efforts in industry and academia. The recent work on e-business, web services, the Semantic Web, rights description and management, privacy, security and trust, is reinforcing the idea of a global approach to deal with the management, interchange and sharing of policies, regulations and business rules between various systems on the Web and in distributed environments.

There are mainly seven rule language technologies including WSML [29], RuleML [19], SWSL [37], N3 [5], SWRL [142], Common Logic and TRIPLE. These are primarily academic efforts; all but RuleML are concerned explicitly with knowledge representation, mainly or only for the Semantic Web.

The RuleML Initiative is proposing the formation of a Policy RuleML Technical Committee and the use of RuleML as a way to interoperate between different Policy systems.

RuleML is a markup language for publishing and sharing rule bases on the World Wide Web. RuleML builds a hierarchy of rule sublanguages upon XML, RDF, XSLT, and OWL. The RuleML project is defining a standard representation for rules of all kinds. The proposed RuleML standard is very broad in its reach: it covers not only forward and backward chaining rules of the sort appropriate for rule engines like Jess, but also transformation and mapping rules and rules that define web services.

Furthermore, the initial goal of the TC is to investigate scenarios of usage of RuleML as an interchange vehicle for policy languages and to develop standards for such interchange suitable for a useful set of such scenario areas [81].

2.5.4 Business Rule Management System

A Business Rule Management System (BRMS) is a software system used to define, deploy, execute, monitor and maintain the variety and complexity of decision logic that is used by operational systems within an organisation or enterprise [79].

The business drivers for the adoption of BRMS are as follows:

- Current software development practice inhibits the rapid delivery of new solutions and even modest changes to existing systems can take too long.
- Accelerating competitive pressure means that policy and the rules governing automated processes have to be amenable to rapid change. This can be driven by new product development, the need to offer customisation and the need to apply business process improvements rapidly to multiple customer groups.
- Personalising services, content and interaction styles, based on process types and customer characteristics, can add considerable value to a business processes of organisation, however complex. Natural dialogues and clearly expressed rules clarify the purpose of and dependencies among rules and policies.
- In regulated industries, such as pharmaceuticals or finance, the rules for governance and regulation will change outside the control of the organisation. Separating them from the application code and making them easy to change is essential, especially when the environment is multi-currency, multi-national and multi-cultural.
- Even in unregulated industries, companies subject to the Sarbanes-Oxley Act are required to make their business processes (and thus the rules that they follow) visible. If such rules are scattered through multiple applications, duplicated (consistently or otherwise) in different places and embedded in procedural code, this becomes a costly and nigh impossible exercise.
- Business rules and processes can be shared by many applications across the whole enterprise using multiple channels such as voice, web and batch applications, thereby encouraging consistent practices.

Using BRMSs should decrease development costs and dramatically shorten development and maintenance cycles.

There are seven typical applications of BRMS technology [62] including:

1) Automating procedures for such things as claims processing, customer service management, credit approval and limit management, problem resolution and sales.

2) Advice giving and decision support in such fields as benefits eligibility, sales promotions and cross selling, credit collection strategy and marketing strategy.

3) Compliance with external and legal regulations and company policy.

4) Planning and scheduling of advertising, timetables and meetings, budgets, product design and assembly.

5) Diagnosis and detection of medical conditions, underwriting referrals, fraud (e.g. telephone or credit card fraud), faults in machinery and invalid/valid data.

6) Classification of customers, risks, products and services.

7) Matching and recommending, including suitable products to clients and strategies to investors.

Moreover, Morgan [26] identifies the following indicators of the need for a business rules management system: policies defined by external agencies, variations amongst organisational units, objects that take on multiple states, specialisations of business objects, automation systems, defined ranges and boundaries of policy, conditions linked to time, the quality manual, significant discriminators, information constraints, activities related to particular circumstances or events, and definitions, derivations or calculations.

In brief, business rules are stated procedures that a business follows to accomplish its work. Business provides many benefits. They capture important and time-changing business policies and store them in a repository separate from the application code. Also, business analysts can modify rules to respond to changing regulations, economic pressure, or customer demand, and rules make applications, and therefore the businesses they support, more agile.

2.6 Summary

The research in this thesis focuses on the software evolution of business rule based legacy systems. The background of this research is that legacy systems are important and valuable. The most important assets of commercial legacy systems (e.g. Enterprise Information System) are business rules and they hide in legacy systems. The technique introduced in this chapter are crucial for developing and evolving software systems, which primarily include SOA, information system, software evolution, software reengineering and business rules technique. They cover a number of areas and made significant advances in legacy system evolution. However, most of the existing legacy system evolution approaches only focus on the technical aspect. They pay too much time on the source code analysis but do not consider the influence from the business aspect. In fact, business knowledge embedded in legacy system could be a very useful tool to help software engineers achieve their target. Thus, the aim of the research in this thesis is to evolve legacy system by making use of the valuable business rules of the systems.

Chapter 3

A Conceptual Framework for Rule-based Web Service Evolution

Objectives

- Leveraging legacy system.
- Defining strategy
- Introducing proposed approach.

Much research has been done to make the reengineering process more effective and efficient. Popular reengineering models include the Byrne Model [2], and Evolutional Model [3], which give approaches that have benefit to the reengineering process but have short-comings as well. For example, it is very difficult to add new requirements in the Byrne Model.

There must be a framework in which to motivate the organisation to understand its business opportunities, its application systems, and its road to an improved target system. This report outlines a comprehensive system evolution approach that incorporates an enterprise framework for the application of the promising technologies in the context of legacy systems.

In order to improve the reusability of legacy applications to participate in dynamic e-business, a strategy is developed to wrap them at the relative stable level such as business object layer. A program comprehension technique will be introduced to help identify the most suitable one. A system modernisation effort evolves an existing legacy system in a disciplined way. This effort consists of three basic processes:

1. Reconstructing one or more higher-level, logical descriptions of the system from existing artifacts

2. Transforming the logical descriptions into new, improved logical descriptions

3. Refining these new and improved logical descriptions to source-level code

The degree to which each of these processes is applied depends on the goals of the modernisation effort. For example, a complete architectural transformation requires reconstructing the existing architecture, whereas a retargeting effort requires understanding only the code structure.

There is plenty of design information and abstract program views that software engineers can possibly extract from code, but software engineers want to extract only the information that is helpful for the maintenance or reengineering task at hand. Therefore, software engineers need to identify proper heuristic rules to map program code to low-level program designs, and from there move further to recovering higher-level design views.

3.1 Strategies

With the ever-increasing push for component-based software, there has a consequential affect to the software evolution field. By proceeding to unfold an end-to-end approach to component-based development, the arduous task of adopting changes on large scale enterprise systems turns to be much easy. Moreover, the component-based system can be further extended to a service-oriented system by implementing each of components as a service provider. Benefit from a loosely coupled relationship, components can be reused freely throughout enterprise systems.

Software component is a single artifact that may physically be deployed across two or perhaps more machines. It extends the object-oriented paradigm with design principles that allow large-scale use of components for business application development. In particular, a business component encapsulates both conceptual (business logic) and physical (source code) artifacts, which express some business semantics, and offers them as third-party products to prospective customers.

The proposed approach is strongly architecture-centric. Although most legacy information systems were developed with different architectural paradigms, they still consist of business components that carry with them corresponding objects. These objects are various by responsibilities and collaborations and define quite different granularity level for components in a layered architecture. By reorganising these legacy components, the target architecture can be built layers upon layers as shown in Figure 3-1. The following paragraphs will address them in order,

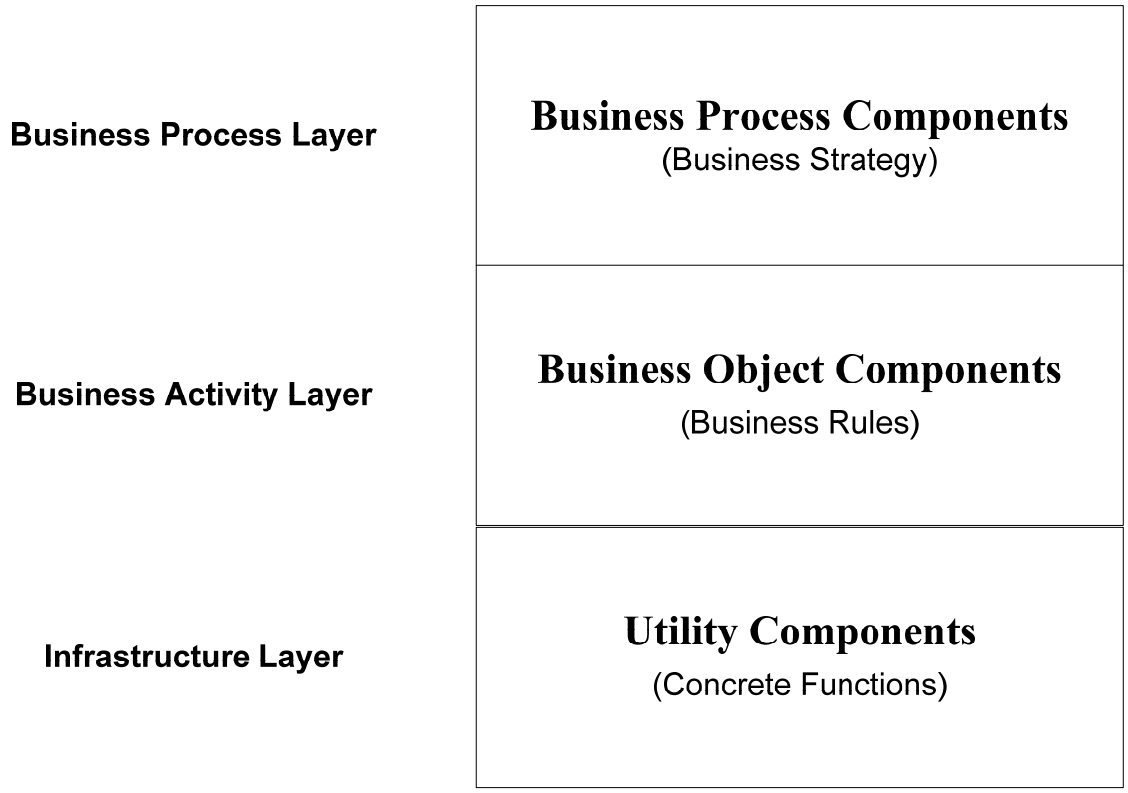

Figure 3-1. Information System Architecture

Utility Component

Definition: utility components are located at the lowest (most fine-grained) granularity in the architecture; provide basic utility services for the whole business system. It is a specific form of the usual concept of component in the industry-for example, it may be implemented as an Enterprise JavaBean, or as a CORBA component, or as a DCOM component. This type of component encapsulates few or no business rules. They are very stable and can be reused by indexing their

functionalities in a look-up table at the infrastructure of systems.

Strategy: Components which provide basic functions or general services are likely to be used as utility components. Commercial Off-The-Shelf (COTS) components and certain legacy components are suitable for this category. As mentioned before, components at this level include few or no business rules; they can be freely deployed at any phases in business process. A standard interface is needed by utility components to provide services across the system. When a utility component is encapsulated by being provided with clean interfaces, the component can be treated as a black box. As such, it forms the largest number of fine-grained components in the proposed system architecture.

Business Object Component

Definition: Business object component is a component that implements a single autonomous business concept. It responsible for business objects management and business activities control. Business object components usually relate with one or more utility components that together address the various aspects of business component. The business object components are relatively stable and tend to have only minor modifications made to them after creation. They are reusable or sharable within particular system. Business rules are usually found in these components.

Strategy: The components used to control business activities in an organisation are better to be built in house rather than bought off the shelf. Because they are usually include rules which are peculiar to the organisation. These components are responsible for manage regulations and objects. The interface to them is basically Create, Search, Read, Update and Delete. The business object component is the main concept around which the whole approach is centered.

Business Process Component

Definition: At the top layer of the architecture is the business process component. Business process component contains the business processes workflow that manages the way business performance. It communicates with a group of business object components that cooperate to deliver the cohesive set of functionality required by a

specific business need. It is less reusable than the other two types because it contains business rules that govern whole business process. These components are less stable. They tend to change quite frequently as the organisation thinks of better and different ways of conducting its business.

Strategy: Business process components control the process of doing business. They are unique to each organisation since the environment and conditions are various from case to case. An easy way to create business process components is to split them into two parts. One part describes the business process such as in which order organisation doing business. The other part tailors the description to a particular business process by providing restrictive business rules. These business rules can be turned into plug-ins and changed flexibly.

3.2 Methodological Framework

The global architecture underlying the methodological framework is made up of three building blocks: reverse engineering, forward engineering, and alignment.

Reverse engineering is concerned with gaining an understanding of legacy programs and repositories, and defining component wrappers that encapsulate them into software layer so that they look and behave like components. As most of the legacy systems are huge and complex, it is wise first identify its useful parts so that the software engineers can put energy on the limited work.

Forward engineering derives a set of business components from the requirements of new business processes. These components are depicted as chevrons in the top tier of the figure.

Alignment of to-be business processes and as-is legacy systems implies that legacy components that have significant overlap with new business requirements (e.g., required services) are mapped to new business components, signifying that these business components reuse the functionality of the legacy components. Remaining legacy components can be ignored. Hence, alignment starts with a thorough understanding of the relationship between business requirements and available

information technology capabilities. One important aspect of alignment is to identify gaps and matches, after which new software may need to be constructed to fill gaps and to integrate with reusable information technology resources. Both issues are addressed in the methodological framework.

3.2.1 Management Architecture

Figure 3-2 depicts a pyramid styled management architecture.

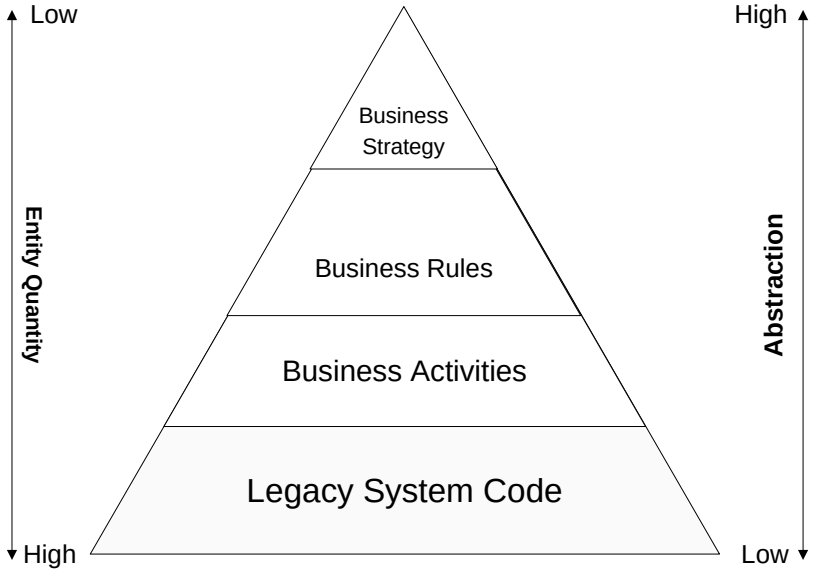

Figure 3-2. Four-layered Structure of an Enterprise Information System

The systems at the bottom are source code of mostly legacy systems such as workflow management systems, inventory management systems, scheduling systems, and databases.

The next level up in the pyramid is the business activity level that interacts directly with the program code and information sources. This level executes control to programs on behalf of the higher level of the architecture, meanwhile report the information generated by the lowest level legacy systems to make further decision.

Business rules level detects the situations as business activities execute, and respond to the situations by rendering management actions back to the execution management layer. The main component in this level is business rules management

system.

The highest level of the enterprise information system architecture is the business strategy layer. It defines the general strategy for operate business. This area covers macroscopic topics such as business goal, market prediction, risk analysis and competition strategy etc.

The flexibility and quantity of entities contained in each level are decrease along with the movement from pyramid bottom to top. In practice, business strategies guide the enterprise goals and directions, which are usually expressed as a collection of abstract descriptions. Business rules reflect these descriptions as concrete instructions. Each rule can invoke related components to achieve specific functions. The components in the information system will then execute program code to perform the activity. This process clearly reveals the relationship between each layer in the enterprise systems. The proposed approach aims to analyse legacy enterprise system according to this architecture.

3.2.2 Methodology

Evolving legacy system is always a painful period. The trick to keep the value of legacy systems is extracting the business knowledge embedded in the systems while at the same time enabling those systems to work with new technology-based systems. For most legacy system, the valuable business knowledge are represented as program operations and are stored as program codes. Success lies in finding a way to make ordinarily inflexible, proprietary systems more flexible and open. The tasks are listed as follows:

1. Evaluate the value of legacy systems and decide what kind of solution should be applied.

2. Understand the overall working process of legacy systems and separate their major functions.

3. Divide legacy systems into segments and regroup them for control over functional management.

4. Extract business logic from legacy source code and represent them as in a standard format.

5. Make target information systems more flexible and give minimal impact on business performance.

6. Lower the technical requirements of operating enterprise information system. By using understandable business language instead of complex technical specifications

7. Business rules management systems like Blaze Advisor have interactive testing, execution flow, cross-reference tools, and reporting features to aid in development, testing, and documentation.

8. Business rules management systems have predefined rule replacement features to handle system updates without interrupting service to application users.

9. Business rules can have explicit times and dates when they should go into and out of effect.

10. Rule management templates can be created to let users update, view, or create rules in a controlled manner without knowing anything about syntax or code.

11. The rule engine can quickly look through large sets of rules, finding the proper ones to apply based on case-specific conditions. No code is required to specify which logic subroutines should be fired in which order.

Program understanding is a prerequisite for legacy system evolution aim at acquiring knowledge about a software artifact through analysis, abstraction, and generalisation. This is usually a complex and expensive process because most of the legacy assets are constructed to run on various obsolescent hardware types and are programmed in obsolete languages.

Several methods for identifying objects in code can be defined depending on the desired level of granularity. The choice of the granularity level is driven by the particular problem which motivated the search for objects in procedural legacy systems.

The research focus on the problem of migrating existing software resources, i.e. programs and persistent data, towards an object-oriented platform to make new object-oriented software able to exploit these existing resources. Therefore, the proposed approach consists of searching for coarse-grained persistent objects, instead of fine-grained volatile objects.

Arranging business rules discovery phase starts after a well-defined scope and purpose have completed can helps reverse engineers targeting the potential candidate rules from legacy systems (i.e., mining rules for a particular sub-function of the application, particular state or region, etc.).

Software engineers regard the business activity meta-model as an appropriate starting point for the business rule refinement process at the business process activity level. The business rules that underline the business activities are first described in a natural language. In subsequent steps, these rules are refined in a structured way as a set of structured rules representing the business process at different abstraction levels.

3.2.3 Evolution Process

The proposed approach aims to extract the reusable legacy assets from the underlying legacy system and re-host them in the new target architecture. In this context, methods and processes are needed to decompose existing systems into notable sets of components, each of which potentially implements an object. However, LISs are huge and usually the packaged systems that composed by rich and old structures. They tend to have have process, objects, rules and utilities mixed together. To address this problem, a specific decomposition method is developed. The whole process can be divided into three main phases which are preliminary analysis, coarse-grained decomposition and fine-grained decomposition. Figure 3-3 presents a summary of the process description, which is expanded in the following sections.

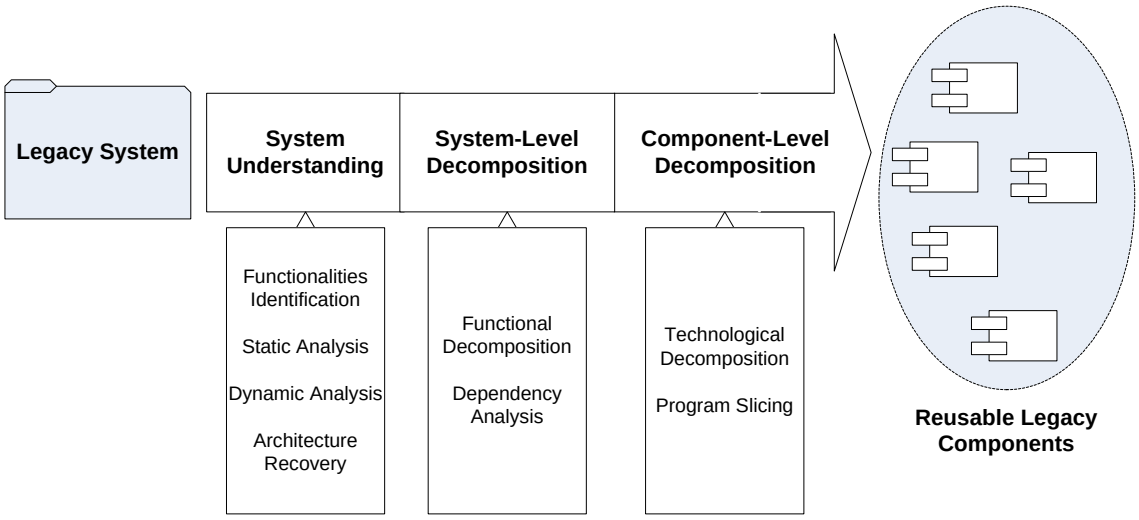

Figure 3-3. Evolution Process

At the top of the figure is a flow arrow diagram (high-level process description) which shows the major activities in the decomposition work. Below the arrow diagram are summaries of the tasks performed in each of the major phases.

Step 1. Preliminary Analysis

The first step implements a large-scale observation on legacy systems by using preliminary analysis. The goal at this stage is not to provide a comprehensive analysis of entire source code, but to discover the general structure of legacy system. Once a general understanding of the layout of the legacy system is established, engineers can then proceed to more in-depth work to determine the areas of special functionalities within it.

Step 2. System-Level Decomposition

Step two encompasses the knowledge of identify components from legacy systems. This phase aims to identify system-level components which are simply a set of cooperating business object components able to provide business solutions. The business object components normally implement one or more part of identifiable and autonomous business processes. They can be standalone or distributed systems but not designed as service orientation. Functional decomposition technique is used in this step to allocate the structure scopes to each

business object component based on their functionalities. In addition, the component dependency relationship is another point that needs analysis in this step. It is a common situation that the isolated components or sub-systems corresponding to a specific functionality is not modulised well and could be entangled with other components in the legacy system. Hence, a dependency analysis is needed to depict a detailed relationship among these components or modules. If such a relationship can be retrieved, service identification can be achieved more easily and efficiently.

Step 3. Component-Level Decomposition

The component-level decomposition follows the work of system-level decomposition to provide detailed information on a selected business object component. Extracting the "pure" business utility components out of the tightly coupled presentation logic is the main task at this phase. Most legacy applications were developed with different architectural paradigms in mind and typically consist of elements that are of a fine-grained nature. They are not designed to be used in SOA. Therefore, what is required is an appropriate technological decomposition, such that the core parts that fulfill tasks in the component are isolated as candidate services or service constituents.

Overall, the proposed approach implements reengineering technique from the most coarse-grained legacy system structure to the finest-grained legacy components structure, each cycle is regarded as an instance of the reengineering model. The advantage of this mixed-strategy approach is improving efficiency and reliability of the decomposition work. It treats the legacy components as a set of separate black boxes, each of which responsible for very specific task and can be reformulated for integration with other systems. Instead of taking an automated brute-force approach to static and dynamic analyses starting from the source code, the black-box approach analysis legacy systems at structural level and identify the reusable components without regarding their internal implementation details. Different programming languages and development platforms approach this concept differently, but the general idea is almost the same and involves these three steps of decomposition.

3.3 Summary

What the above discussion makes clear is that the componentised transformation and service-oriented migration could combine together to achieve the target of improving value and evolvability of legacy system. Considering the specific characters of enterprise information system, engineers need to pay more attention on the business knowledge migration. Business knowledge is the most important asset to legacy systems. Its concept can include from long-term organisation strategy to very detailed operation regulation. Hence, the integrity and functionality of business knowledge are the vital to the success of legacy systems evolution. However, since most of the legacy systems have been developed over a long time, their business knowledge is spread across the whole system. To cope with this problem, other than only decompose legacy systems like traditional methods; the proposed approach makes a further step to divide the decomposed legacy elements into different level based on their responsibilities. Each level of granularity corresponds to one specific category of software component and associate with one specific strategy to manipulate.

Chapter 4
Legacy System Understanding and Partitioning

Objectives

- Legacy System Understanding and Architecture Recovery.
- System-level Partitioning towards Domain-specific Business Knowledge.

The proposed architecture recovery approach is based on functionality identification. By searching for functionalities that implement the complete business applications, the component identification process can be very effective and efficient. Then the identified components are analysed by using a mixed static and dynamic modeling approach which supports legacy system architecture recovery.

Then, based on the derived Module Dependence Graphs (MDGs) and the understanding of domain knowledge, a specific GAs-based clustering approach is given to partition modules which are generated during architecture recovery.

4.1 Tradeoffs Between Business and Technical Requirements

Legacy system is by definition a successful system with long term maintenance. Because most of them were constituted by obsolete techniques and hardware, the portfolio of legacy systems needs to be critically assessed and evaluated periodically to ensure the applications are still aligned with new business needs and technical requirements.

When a new requirement is detected, engineers will apply small patches to keep the legacy system up and running. However, such type of maintenance may not be the most cost-effective solution when the changes are related with the core functionality of legacy systems. Upgrading or replacing a complex system that is supporting a critical piece of business is neither easy nor cheap. Business and IT change are involved, and the trade-offs are complex. Therefore, before executing the legacy systems maintenance, every engineer should concern the following question: “Is it worthy to maintain this legacy system?”

Figure 4-1 shows the matrix to help reverse engineers make decisions about which strategy is suitable for different situation:

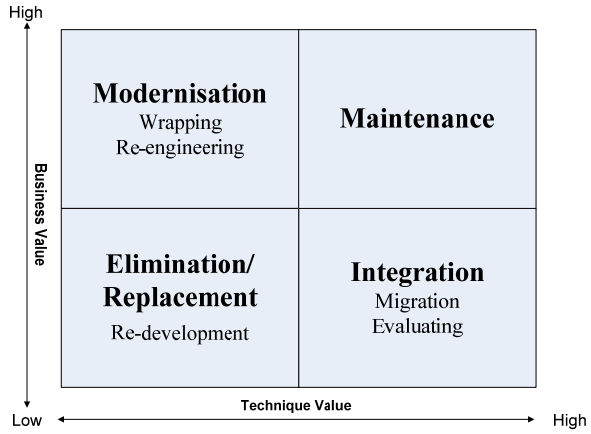

Figure 4-1. Discipline of Manipulating Existing Systems

The diagram evaluates legacy systems in a portfolio along two orthogonal dimensions. On the X-Axis is plotted the Code Quality of the existing legacy application and on the Y-Axis, is plotted the Business Profit, they all varies from Low to High. Each dimension is demarcated into two regions reflect as high and low respectively. Take account of both segmented dimensions, a four quadrants portfolio matrix indicates four specific strategies. Each strategy involves certain trade-off that needs to be analysed and makes the most sense of legacy system evolution.

Elimination and Replacement: From a business perspective, chasing profit is an eternity theory that has never change. It is not wise to invest in a legacy system that cannot bring enough profit. Furthermore, if the techniques used in this part of legacy

systems are fail to meet state of the art technical requirements, it is the time to consider the possibility of retirement or replacing. Because the business value produced by legacy systems in this quadrant is relatively low, system replacement is preferably implemented by purchasing low-cost packaged software like an ERP system or commercial off the shelf components.

Integration: Although the technical condition of the systems in this quadrant is considered relatively high, they generate limited profit to the enterprise. Management should re-evaluate them and determine whether they could be strategically repositioned to support the more value-adding business processes. These systems are the most likely candidates for integration with high technology based system to extend their usage. Alternatively, these systems could be left intact and reassessed in due time.

Modernisation: In practice, great deals of legacy systems fall into this quadrant. The legacy systems capture substantial business value but suffering from a poor technical condition. As such systems are getting difficult to add new functionality and keep up with business requirements, most companies want to transform them into technically superior enterprise systems or encapsulate and integrate them with new, value-adding applications. However, while trying to move to this arena, many companies are stuck with these large, heterogeneous and mission-critical systems. It is common place that these systems are tightly linked with the business workflow and have business rules and policies integrated in the code. Therefore, many companies regard modernisation as somewhere between improbable and impossible.

Maintenance: Enterprise systems whose values fall into this quadrant are crucial to the success of an organisation. Enterprises operating in this situation can generate high business value and operating as technically sound systems. It is recommendable to keep and maintain the existing system till costs become an over-riding factor to shift.

Take a look of all these four strategies, the most complex problem of legacy systems is that they aim at supporting business but are implemented based on outdated techniques. Thus, resolving problems of legacy system evolution falls in modernisation. Therefore, the discussion of this article will focus on the modernisation strategy, and discuss three aspects including reengineering legacy systems, recovering business

knowledge and migrating to advanced platform.

4.2 Legacy System Understanding

Legacy system decomposition starts with system understanding. System understanding is a prerequisite for legacy system evolution. It can be regarded as a deductive process of acquiring knowledge about a software artifact through analysis, abstraction and generalisation. Understanding the architecture of legacy systems and extract the right components is critical.

The critical activities in this phase include identifying business objects and recovering global structure of legacy systems. These tasks require business experts to mine the existing systems for valuable functionalities and with the assistance of the system engineers to establish a general understanding of the underlying architecture layout. A thorough understanding of potential reusable components mapped onto business requirements is generated as an output of this stage. It will facilitate the process of decomposing legacy system at a high-level perspective in the next part of this chapter. The legacy system understanding process is structured as a sequence of activities in the following sections.

4.2.1 Functional Model Identification

System evolution and technology insertion do not take place in a vacuum. Evolution happens in the context of an organisational setting that is varies considerably in terms of the culture and the readiness for adopting changes. Many attempts at evolution and migration failed because they concentrate on a narrow set of technology issues without considering the broader set of management and business issues. While there may be many complex technical problems that are largely unprecedented, a focus on the technical problems to the exclusion of the business and organisation problems is still not a perfect solution. In the business context, the most effective way to judge of a successful evolution is to examine the performance of business services from the aspects of stability, agility, continuance and efficiency. All these aspects reflect to the functionalities of components. Hence, it is crucial to execute system evolution on the

premise of understanding which legacy functionalities are compatible with the requirements of high-performance target system. Generally, the process of functional model identification can be illustrated in Figure 4-2.

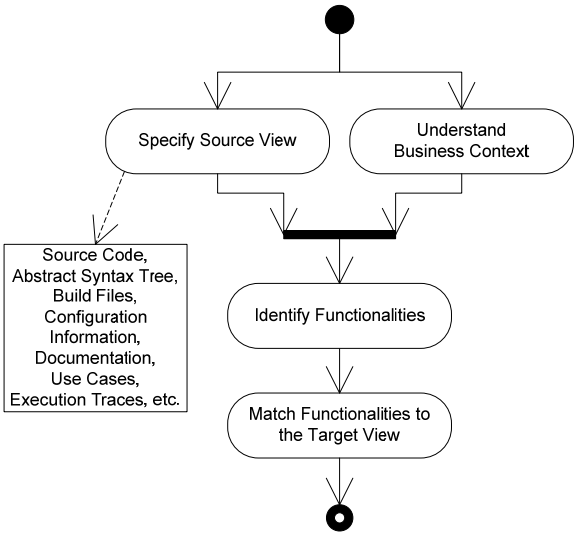

Figure 4-2. Process of Functional Model Identification

The objective of functionality model identification is to identify and document functionalities in legacy systems. During the process, the analysis primarily targets on the valuable legacy functionalities that may be considered for inclusion into target system.

From this understanding, the process of Functional Model Identification can be divided into two parts: Information Collecting and Reusable Functions Selecting, in which the former one is composed of specifying source view and understanding business context, and the latter one refers to identify functionalities and match functionalities to the target view.

1. Information Collection

Developers often encountered a situation that there is very little known about the technical details of legacy system from the original development team. It turns out that one of the crucial tasks at this stage is collecting information and documentation from very valuable but incomprehensible codes. The following is a list of relevant activities

to accomplish this task:

- Discuss with Maintainers.
- Skim the Documentation
- Observe System Operation
- Brief Review Codes
- Study Use Cases

There is often a strong parallel between system architecture and its underlying business concept model. A description of legacy system structure and behaviours from use cases will reveal various functionalities which are corresponding to different components. Use case can also be used as a means to translate higher-level systems engineering requirements and functional artifacts to a meaningful set of information for software developers. In this way, the “shall” statements in the higher-level requirements can be mapped to use cases as a simple means to show traceability. The elaboration of these use cases can provide the necessary information on how the software subsystems and components will communicate.

In certain situations some activities are not applicable due to a lack of resources, such as the maintainers have left, documentation becomes inconsistent etc. This is not necessarily a problem because some of these activities may be irrelevant for the goal of reengineering project. However, the absence of resources is a potential risk to the project, and it should be recorded as such in the project report. Developer should plan to keep the report up to date while the reverse engineering project progresses and the understanding of the legacy system grows.

These activities should be applied during a short time span. After this period developers should grasp the main functionalities and plan further activities based on that knowledge.

2. Reusable Functions Selection

Once the necessary functionalities information has been obtained, it is time to

compile the first project plan – Reusable Functionality Selection. Rather than traditional evolution approaches that are based on module division, the proposed approach is developed from functionality division. That is means system is regarded as an aggregation of functionalities. Correspondingly, legacy systems will be analysed and divided into a list of functionalities.

In practical scenario, five situations are commonly occurs during the transformation from legacy functionalities to target services. They are called: One-to-One, Many-to-One, One-to-Many, Creating and Discarding. All of these five situations are represented in the following equation:

$$TS = \sum_{i=1}^{n} s_i = \begin{cases} \sum_{i=1}^{n} f_i & if\ s_i = f_i \quad (1) \\ \sum_{i=1}^{n} f_i - \sum_{i=n_1}^{m_1} f_i + s_i & if\ s_i = \sum_{i=n_1}^{m_1} f_i \quad (2) \\ \sum_{i=1}^{n} f_i - f_i + \sum_{i=n_2}^{m_2} s_i & if\ \sum_{i=n_2}^{m_2} s_i = f_i \quad (3) \\ \sum_{i=1}^{n} f_i + \sum_{i=n_3}^{m_3} s_i & if\ \sum_{i=n_3}^{m_3} s_i = new\ services \quad (4) \\ \sum_{i=1}^{n} f_i - \sum_{i=n_4}^{m_4} f_i & if\ \sum_{i=n_4}^{m_4} f_i = redundent\ functionalities \quad (5) \end{cases}$$

TS: Target System s_i: Target Service

1) One-to-One: In this situation, the functionalities implemented in legacy system are exactly corresponding to the services in the target system. All of the legacy functionalities are reusable and act as independent services in new system.

2) Many-to-One: This situation means that one service in target system is equivalents to more than one functionalities work together in legacy system. It helps to make a clearer structure of target system and improves the efficiency by wrapping a bunch of simple functionalities into a comprehensive service.

3) One-to-Many: This situation is opposite in appearance to “Many-to-One”. It happens when functionalities that compose a complicated functionality in legacy system are reusable to target system. By transferring these basic

functionalities into separate services, the value of legacy system is maximised.

4) Creating: In most of the situation, engineers need to add new services into evolved systems to meet the emerging business requirements.

5) Discarding: Correspondingly, as the business requirements changing, some old features or functionalities become useless and abandoned during system evolution.

The process that leads to identifying functionalities is iterative and involves business and software engineering knowledge. Business experts aware of application domain concepts that are candidates for target business services, and software engineering experts judge which design and implementation concepts are candidates for sound and useful business utility components.

Functionality identification produces a functional description to legacy systems. Such a description is quite rough and is mainly concerned with the behavior of the legacy systems and not with their structures. However, it may serve as an ideal initial hypothesis to be further refined by applying speculates about architecture. The identified legacy functionalities will be highlighted for closer study as potential inputs to the target system. Most importantly, the equations that represented in this section provide a common basis for legacy functionalities mapping target systems and a taxonomy basis for future discussion. With the functionalities identification, legacy functionalities can be evolved separately from the overall system.

4.2.2 Static Analysis for Hierarchy Viewpoint

The rough understanding of functionalities in legacy systems gives reverse engineers a clue to find which design concept should be applied on each implementation. Static analysis is a suitable method to compute potential classes playing a certain role in a design pattern.

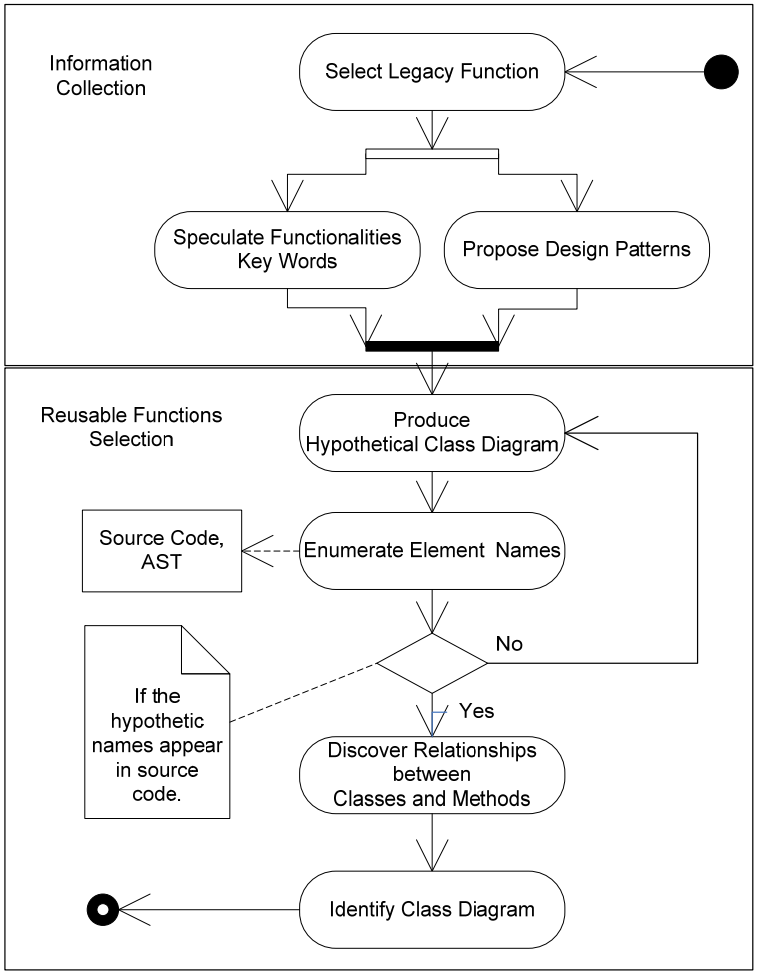

Figure 4-3. Class Diagram Identification

In a structural system, a complete function is implemented by a group of procedures. Object-oriented languages provide a well defined interface to express such complete functions as objects. Each object contains classes that implement a set of responsibilities, and these classes should match against the design pattern proposed by system developers. Based on this idea, reverse engineers can conceive a hypothetical class diagrams to specify the structure of objects and refine them gradually by searching the legacy code. The refinement process first verifies whether the names of candidates in the hypothetical class diagram occur in the legacy code or AST nodes; and then adapts the hypothetical model accordingly. This process will keep repeating until the obtained class diagram is stabilised. The hierarchy view is derived from these class diagrams to describe the decomposition, layering, and generalisation of modules and their use

dependencies. Development experience and the knowledge of legacy system are used during this process. Figure 4-3 shows the process of Class Diagram Identification.

Generally, the static analysis process can be summarised as five steps:

1) Based on the understanding of the legacy functionalities and business context, reverse engineers computes the pattern relation on a specific reusable functionality and provides the result as a set of candidates, i.e. a design pattern instance, with the appropriate static structure. The pattern instance is defined by a set of program elements such as classes, methods or attributes conforming to the restrictions or rules of a certain design pattern. This set is a conservative approximation to the actual patterns in the code. Analysing with the pattern static restrictions can limit the amount of candidate occurrences in the code.

2) According to the design pattern instance, a class diagram is developed and serves as an initial hypothesis of what to expect in the source code. Since legacy systems are usually huge, methods and tools are needed to discover the allocation of domain functionalities in source codes. For the names of the classes, methods, and attributes, reverse engineers can make a guess based on experience, design pattern restriction and potential naming conventions. For example, using the noun phrases in the requirements as the initial class names and the verb phrases as the initial method names etc.

3) Enumerate the names in the class diagram (that is, names of classes, attributes, and method) and try to find them in the source code. And then iteratively adapt the class diagram based on the mismatches until reverse engineers obtain a class diagram that is satisfactory. The adaptation includes four kinds of activities:

   - Renaming, when reverse engineers discover the class names used in the source code do not match with hypothesis.

   - Remodeling, when reverse engineers find out the source code representation of the design concept does not correspond with what they have in their model. For instance, transform an operation into a class, or an

attribute into an operation.

- Extending, when reverse engineers detect important elements in the source code that do not appear in the class diagram.

- Replacing, when reverse engineers cannot find the design concept in the source code. This may entail trying synonyms when there are few mismatches but may also entail defining a completely different class diagram when there are lots of mismatches.

4) Use the proposed design pattern as the method declarations form, iterates over all detected classes and their methods. Take an Observer Pattern for example, its method declarations form is `(S.addListener, S.removeListener, S.notify, L.update)` where `S` is the class declaration of the subject of observation and `L` is the class or interface declaration of the corresponding listeners. For each method `m` of a class `c`, first assume it plays `addListener` or `removeListener` role. Consider each parameter type `p` of `m` as a potential listener class and the provided `p` is neither same to nor a super or a subclass of `c`. Next, determine all function calls issued from inside methods `n` of class `c` to some method `u` defined in the potential listener class `p`. The methods of class `c` containing the calls to `p.u` are considered potential `notify` methods and the method `p.u` as `update` method. Use the predicate `isNotifyListener(c.n, p.u)` to test whether method `c.n` might be a notify method. The satisfied condition is `c.n` calls `p.u` and `p` is not a parameter of `c.n`. The result of the iteration is a set of method declarations `Y: (S.addListener|removeListener,S.notify, L.update)`.

5) Iterate over `Y` to generate final candidates of functionalities that match the proposed design pattern. For Observer Pattern, elements of `Y` are iterated for the purpose of connecting corresponding `addListener` and `removeListener` into one candidate. If `S.addListener|removeListener` is an `addListener` method by satisfying `isAddListener(l)`, all methods that own the same `notigy` and `updaty` are considered as its `removeListerner` candidates. Moreover, if there

is no `removeListener` method, null is used to occupy the position of `removerListener`.

The static analysis algorithm of Steps four and five are summarised as pseudo-code in List 4-1.

```
Static Analysis Algorithm
 C = ∅;
 for each class c do
   Y = ∅;    // intermediate result
   for each method m in c do
     for each parameter type p in m where p ≠ c ∧ c ⊄ p ∧ p ⊄ c do
         for each call from c.n to p.u do
           if isNotifyListener(c.n, p.u)
              Y = Y ∪ {(c.m, c.n, p.u)};
   for each (c.l1,c.n1,p1.u1) in Y do
        for each (c.l2,c.n2,p2.u2)in Y where c.n1 = c.n2 ∧ p1.u1
   = p2.u2. do
         if (isAddListener(c.l1))
            if (c.l1 = c.l2)
               C = C ∪ (c.l1, null, c.n1,p1.u1)
            else
               C = C ∪ (c.l1, c.l2, c.n1,p1.u1)
```

List 4-1. Static Analysis Algorithm

After this process, reverse engineers will obtain a class diagram representing the intra-component communications. However, the static analysis itself is not adequate for a complete system understanding. *Firstly*, the class diagram only provides a static view of the class hierarchy. It is not possible to statically discover the real methods and related components that called in a call instruction referring to an interface or class. *Secondly*, the approach uses design pattern restriction and other methods to limit the set of candidate occurrences in the code. In practice, sometimes this set is still large and reverse engineers hardly want to screen all of them to detect the actual instances. Considering these difficulties, it is necessary to step through the dynamic model analysis to learn what components are instantiated at run time and how they interact.

4.2.3 Dynamic Analysis for Component and Connector Viewpoint

The component-and-connector (C&C) view is considered one of the most important ones for the developer as well as for the architect [52]. The component-and-connector

viewpoint enables the representation of software architecture from the point of view of its components, the principal unit of runtime interaction or data storage, and its connectors, the interaction mechanism among components and the "data flow" among them [85].

Component and Connector Viewpoint illustrates runtime behavior described in terms of components and connectors. Dynamic analysis provides a mechanism for C&C Viewpoint to discover how various runtime entities will communicate and interact. This is similar to using static analysis to show the relationship between classes within a design pattern instance, but the C&C Viewpoint is at a higher level of granularity. Additionally, when considering object-oriented software which employs a lot of polymorphism and dynamic binding mechanisms, dynamic analysis becomes an essential technique to comprehend the system behavior, object interactions, and hence to help recover system architecture.

In practice, the candidate class sets conforming to the conservative static analysis is large. Dynamic analysis takes this as its input; execute the program under investigation and monitor the executions of the candidate instances found by the static analysis with respect to the dynamic restrictions. Through this method, only actual interacted components are retrieved.

Taking into account the occurrences of interaction types (classes and interfaces) in use case realisations, a set of common task scenarios are executed and the execution profiles, for example the function call frequencies from different calling functions to each called function, are captured. In this work, dynamic analyses observe which routines are executed when the function of interest is executed and further tracks its chain reaction to discover how components are interacted at run-time. By inspecting the trace execution, reverse engineers can detect visual patterns of the function execution.

The interactional relationship in dynamic environment is recorded in a C&C library, which includes the follow elements and is illustrated in [102].

- Components are principal processing units that can be composed together to form applications. Components encapsulate much of the complexity of the

software inside a black box and provide services through well-defined interface, port and interaction protocols to other components.

- Ports are interfaces through which components interact. Specifically, ports provide procedural interfaces that can be thought of as a class or an interface in object-oriented languages, or a collection of subroutines, or a module in a language. Components may provide ports, meaning they implement the functionality expressed in the port, or they may use ports, meaning they make calls on that port provided by another component.

- Connectors are interaction mechanism for components; typically contain interfaces (roles) and formal behavioral description. They are responsible for connecting uses without exposing the implementation details of components.

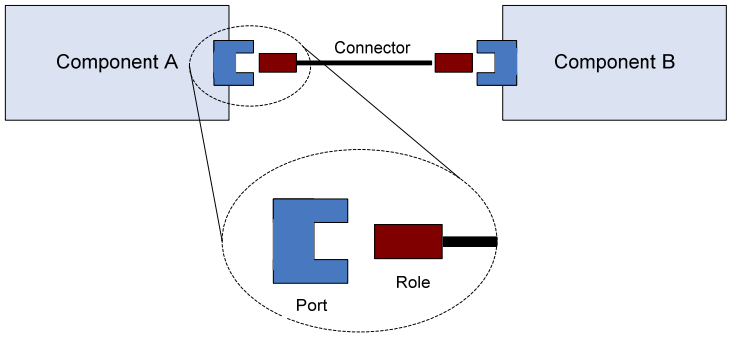

Figure 4-4. Interactional Relationship between Components

In the mean time, reverse engineers aggregate the components to form a dynamic model. Several aspects of a component design should be considered to determine if the dynamic model produces effective component behavior and interactions at runtime.

The dynamic relationship of components is modeled by using component-level sequence or collaboration diagrams to show how specific components interact. These views are especially critical for architecture recovery, as they document the runtime component structures and interactions. The process of dynamic analysis for runtime relationship is shown in Figure 4-5. Once the component and connector viewpoint of the legacy systems is obtained, Architectural Pattern analysis will be applied to recover the system architecture.

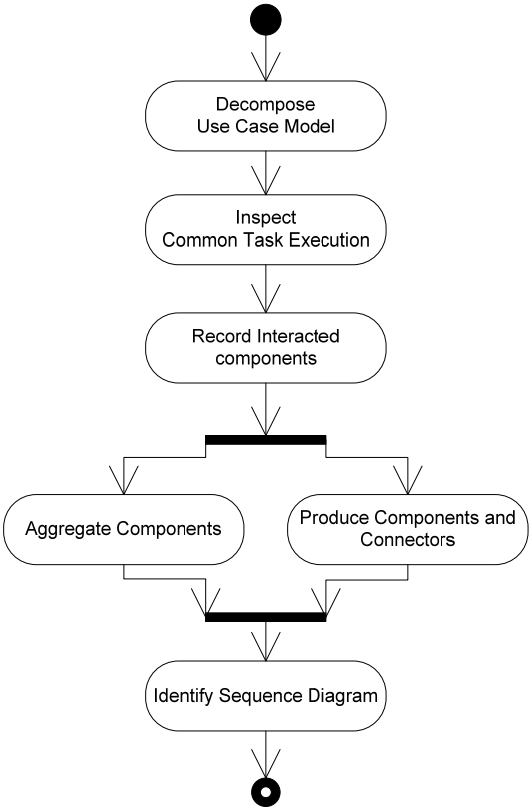

Figure 4-5. Dynamic Analysis for Runtime Relationship

4.2.4 Architecture Recovery

Although the selected viewpoints define the vocabulary and semantics for the representation, they do not define how to present the information. A particular problem of software architecture is the need to understand a combination of multiple views. The identified components represent functional, relatively independent parts of the legacy system. They are good candidates to be used as the architectural elements to build the final legacy system architecture.

Architecture recovery aims to produce a highest level of abstraction of a legacy system. The initial architecture, which is comprised of identified components, should match against an architectural pattern proposed by domain experts. An architectural pattern is an established tool which provides a system template in a specific domain. Many architectural patterns were developed by previous researchers to represent various architectural characteristics. Generally, these architectural patterns can be classified into

six categories by the implementing domain in which they are most prominent. The detail of the category is shown in Table 4-1.

Domain	**Patterns**
Embedded Systems	Pipes & Filters, Client-Server, Presentation Abstraction Control, State Transition, Explicit Invocation, Shared Repository, Layers
Information and Enterprise Systems	Shared Repository, Layers, Model View Controller, Presentation Abstraction Control, Broker
Web-Based Systems	Broker, Layers, Pipes & Filters, Explicit Invocation
Dataflow and Production	Layers, Shared Repository, Presentation Abstraction Control, Broker, Plug-in
Games	Model View Controller, Pipes & Filters, Blackboard
Scientific Applications	Pipes & Filters, Shared Repository, Client-Server, Presentation Abstraction Control, Blackboard, Layers, Plug-in

Table 4-1. Various Patterns Match to Specific Domain

Domain experts have experiences in certain domains and know the most appropriate patterns for that domain even if it is only implicitly. However it is important to make this knowledge explicit: to map the architecture patterns to the domains they are most frequently used, and to reason about this mapping. The proposed architectural pattern gives a clue to the search of candidate architecture, whilst the discovered relationships between components are used in the next step to refine the initial architecture.

Every software system exhibits certain characteristics that are reflected in source code structures or elements. Based on the proposed pattern category, legacy system architecture can be abstracted by refinement of pattern definitions and aggregation of coherent architectural elements. Figure 4-6 illustrates the process of recovering architecture from Pattern Matching.

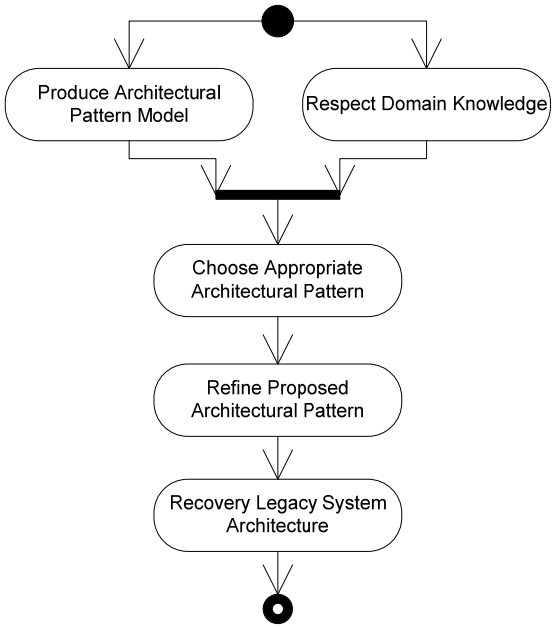

Figure 4-6. Recovering Architecture from Pattern Matching

Architectural patterns are represented as a group of C&C records which refers to patterns of domain with predefined beliefs. During the process of architecture recovery, an architectural pattern is matched against the C&C library which is generated by dynamic analysis. Every successful matching between the records and C&C library may lead to revision to the beliefs of the relevant records in the final C&C diagram. An architectural pattern will be recovered when the related beliefs reach a predefined level. At this point, all the components participating in this architectural pattern can be grouped by their respective roles to produce a complete architecture description.

The output of this refinement process is a hyper-structure offering a holistic perspective on the software system as a foundation for investigating the concrete impact of architecture on the problems signaled. This hyper-structure includes traceability links between views and links to other software artifacts, such as subsystems, components and classes etc. The ideal recovered architecture will allow reverse engineers explore legacy systems at various levels of abstraction: it lets them zoom in and zoom out between sources and structures and navigate between views.

4.3 Legacy System Partitioning

Legacy systems in real world are usually huge in size, ranging from tens of thousand of Lines of Code (LOC) to millions of LOC. As learned from Section 4.1, a legacy system is often composed of modules which implement functionalities in different domains. Obviously, it is useful to break a description of a complex system down into a set of small, manageable subsystems, each of which contains modules that are most relevant. The activity of separating legacy systems into self-contained subsystems depend on the discrepancy of domain knowledge embedded in each module.

Domain knowledge partitioning happened in the early stage of legacy system reengineering, which involves defining domain knowledge granularity, clustering modules, classes and components into subsystems for further processing.

4.3.1 Domain Knowledge Validation

While recovered legacy elements do much to improve software development and maintenance, they are insufficient for supporting the design and ongoing maintenance of large legacy systems. Such systems often contain several hundreds of thousands of LOC that are packaged into a large number of cooperating modules. Fortunately, these systems are often organised into identifiable clusters of modules, called subsystems, which collaborate to achieve a higher-level system behavior. However, the subsystem structure is not obvious from the source code structure. A method is needed to create a hierarchical view of the system organisation.

The idea of identifying subsystems is similar to the previous work that grouping related procedures and their associated data into modules (or classes) based on the functional model. But reverse engineers need to consider the standard of grouping modules from a higher level. The choice of the granularity level is driven by the particular problem which motivated the legacy systems evolution. According to the goal of extracting business rules, the suitable granularity for partition legacy system can be summarised as the extent of business logic discrepancy that embedded in various subsystems. Modules that contribute to the same business logic are included in a

subsystem to express specific domain knowledge of legacy system. Analyse over all modules will finally results in a collection of subsystems, each of which responsible for a complete business role.

Subsystems help developers navigate through the numerous software components and their interconnections. The separation of these subsystems represents the granularity of the high-level decomposition in legacy system. Figure 4-7 shows the desired legacy system structure after domain knowledge partition.

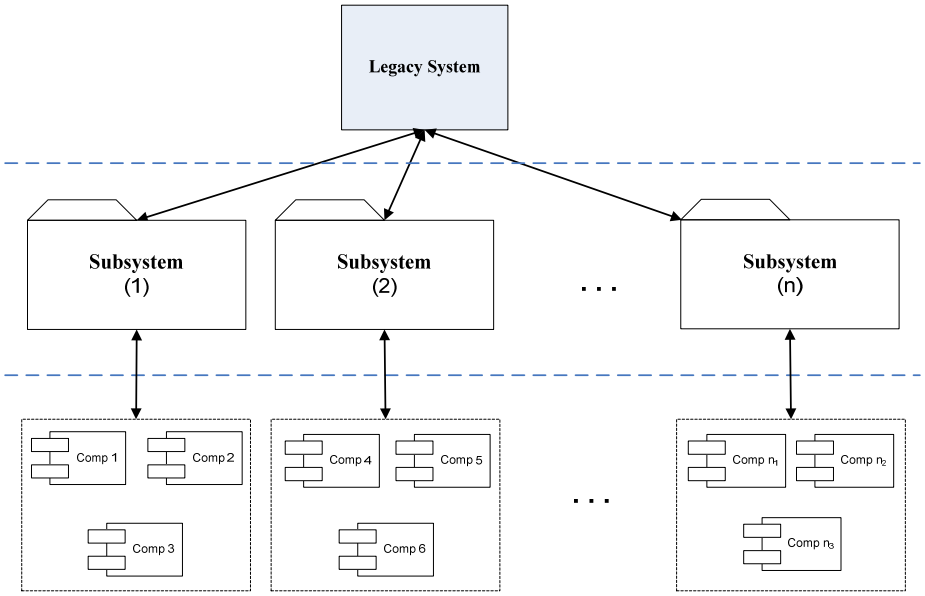

Figure 4-7. Desired Legacy System Structure

Based on this understanding, the flowing sections will concentrate on searching and investigating business logic among legacy elements.

4.3.2 Module Dependency Analysis

Domain knowledge validation defines a functional scope to every potential subsystem and provides a guideline to aggregate related architectural elements, but it is still not enough for executing legacy system partitioning. When reverse engineers want to extract a component to build subsystem, they must first investigate which components will affect the extracted component and which components will be affected by the extracted component. Although the legacy system architecture has been

recovered in Section 4.2.4, it shows only a hierarchy between architectural elements but does not clearly expose their dependence relationships. The change impact analysis is therefore necessary to legacy system partitioning.

Most legacy systems consist of a large number of interdependent elements, such as classes, procedures, data structures, variables and binary modules. These elements are often heavily coupled between each other in intricate ways, like procedure calls, inheritance relationships and variable references. In such situation, modification at one program segment will often affect other modules in unexpected and undesirable ways. Therefore, the strategy of coping with legacy system evolution should not only focus on system partitioning, but also consider the relationship between the functional elements and their technical environment.

Tracing the impact of change in a legacy system is a challenge, and may take up a significant portion of the overall maintenance effort. In order to precisely decompose legacy systems, it is most helpful to show how modules depend on each other by building a Module Dependence Graph (MDG), where modules refer to components generated from Section 4.1, and MDG represent the dependencies among components.

Dependency analysis is utilised to show high-level dependencies among modules. The structure of the finished systems can be represented as a MDG where the modules (e.g. files, classes) are represented as nodes and their relationships (e.g. function calls, inheritance relationships) as directed edges that connect the nodes. Being able to classify each unit as either dependent or independent, by itself, is not a premise of a perfect partitioning work. However, a MDG can be viewed as an indicator of constructions borders. Moreover, it can further be viewed as an indicator of dividing the whole system into organically integrated parts. With the help of dependency diagram, legacy system can be partitioned into subsystems that have high cohesion value according to their module dependency relationships.

If used properly and developed in parallel with the program, MDGs are useful aid in checking for “knock-on effects” of any changes in lower level units, and a method of measuring how well structured a program is. Because its weakness in illustrating the structure of system architecture, it is recommend using dependency analysis in

conjunction with established qualitative judgments such as cohesion.

When one module cannot occur until another module is completed, there is a dependency between them. In a legacy system, all modules can be classified as either dependent or independent. The dependent modules invoke at least one other program to aid their operations, but the independent modules do not.

Take a shopping process for example. The sequence starts from the customer input module, which triggers a function call to list all selected items. As calculating the total price, the price calculator should consider many factors that are vary from customer to customer. The most common factors include the quantity of purchase, address and delivery date. Customer can use payment module to get the total price from calculator. Once the payment is clear, the delivery could be process.

The dependency relationships between modules can be tracked through the use of a dependency table, which lists every unit, followed by its dependent units. The dependency table of the previous example could look like Table 4-2.

Unite	**Dependent Units**
Customer Input	Calculator, Delivery
List	Customer Input, Calculator
Calculator	Payment
Payment	Delivery
Delivery	-

Table 4-2. Dependency Table of Price Inquiry Component

Dependency table clearly illustrates the dependency relationships between each unit and helps to draw MDG. First place each unit shown in the dependency table on the diagram, and then, link them according to the relationship shown in the table. It is much easy for programmers to draw a complex MDG as Figure 4-8.

Reverse engineers can extend this dependency analysis approach to other application domains. Since the legacy system architecture has been recovered already, and the scopes of elements corresponding to different legacy functionalities were clear, the work of module dependency analysis could be very easy and feasible to apply on all other legacy functionalities.

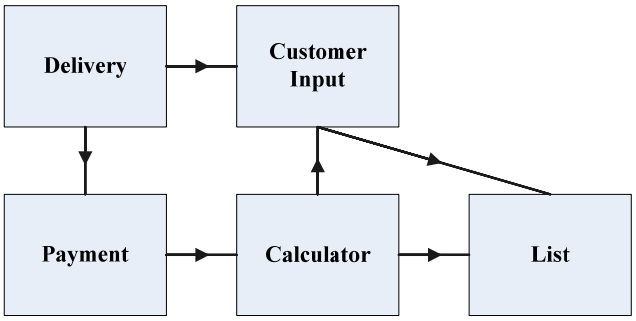

Figure 4-8. Module Dependency Graph of Price Inquiry Component

Although MDGs are excellent for exposing module relationships, it is not necessary to display the entire dependencies for whole system. Legacy systems are very likely to outgrow the graphical model. As the complexity of systems keep increasing, more nodes and edges are added to the diagram, this lead to the diagram gets less readability. Instead of charting the entire system, it is often a better idea to partition them by grouping closely related nodes into clusters (i.e. subsystems), in what is known as the software clustering problem.

4.3.3 GAs-based Clustering Analysis

Hierarchical or partitional clustering are two traditional methods that are widely used nowadays. In contrast to hierarchical clustering, which yields a successive level of clusters by iterative fusions or divisions, partitional clustering relocates entities by moving them from one cluster to another, starting with an initial set of cluster centers, chosen at random or according to some heuristic procedure. Considering Section 4.3.1, in which the desired object subsystems have already been pre-set, partitional clustering is more adequate at this phase than the hierarchical clustering. List 4-2 presents the pseudo-code of the partition clustering algorithm.

```
Partitional Clustering Algorithm (S, K)
Where:
S - Entities Set
K - Number of Clusters
Initialise K cluster centers;
WHILE (termination condition is not satisfied) DO {
    Assign entities to the closest cluster center;
    Update cluster centers using the entities assignment;
}
```

List 4-2. Partitional Clustering Algorithm

However, the drawback of the partitional clustering is obvious too. It implements optimisation algorithms (most commonly the *K*-means algorithm) to execute an exhaustive enumeration process of every possible partition throughout the MDGs and chooses the best one. Although such brute force method guarantees to reach the optimal solution, it is infeasible in practice due to the extremely expensive computation.

Given a set of entities $x_i \in \Re^d$, $i = 1, 2, ...N$, partitional clustering algorithms aim to organise them into *K* clusters $\{C_1, C_2 \ldots C_K\}$. The permutations and combinations formula given by G. Liu in his book "*Introduction to combinatorial mathematics*" calculates the amount of all that possible partitions as below:

$$SUM = \frac{1}{K!}\sum_{m=1}^{K}(-1)^{K-m} C_K^m m^N$$

Obviously, even for a small scale clustering problem, simple enumeration is impossible. The recurrence equation grows exponentially with respect to the number of nodes (*n*) in MDG are partitioned into *k* subsystems ($1 \leqslant k \leqslant n$). For example, in order to group 30 nodes into three subsystems, the number of possible partitions solutions is approximately 2×10^{14}.

To efficiently explore the extraordinary large solution space of all possible partitions of the given MDG, Genetic Algorithms (GAs) are applied. Inspired by the natural evolutionary process, GAs search the large solution space by using a set of evolutionary operators and converge into a globally optimal clustering. In GAs terminology, each *string*, representing one candidate solution, is an encoding of input data of the problem, also known as a *chromosome*. A collection of such strings is called

a *population*. Initially, a population is generated either randomly or according to some heuristic rules, which represents different points in the solution space. An optimisation function, called the *fitness function*, is associated with each string to evaluate the degree of goodness of the string. Population then can be evolved iteratively via the processes of *selection*, *crossover* and *mutation*. Each iteration of the process is called a *generation*, as depicted in Figure 4-9.

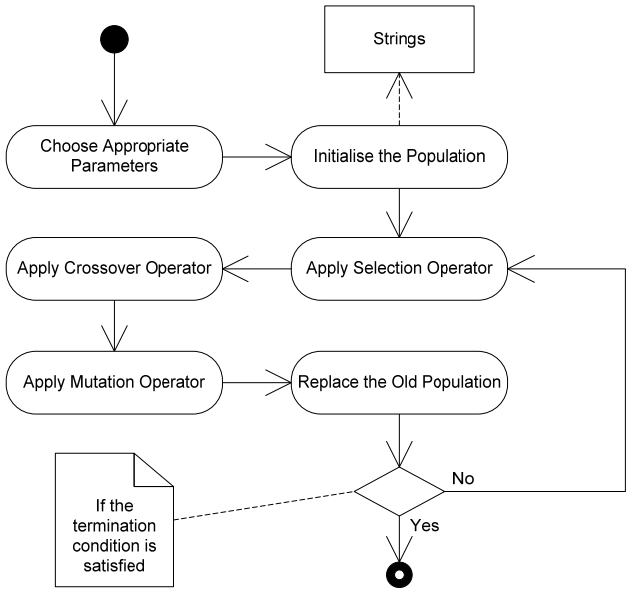

Figure 4-9. The Process of GA-based Clustering Analysis

Generally, in the context of GAs-based clustering, each chromosome in the population encodes the information for a valid partition of the entities. These partitions are altered with evolutionary operators; the best ones with high scores of the fitness function are retained. The selection operator ensures the continuity of the population by selecting the best strings in the next generation. The crossover operator cuts two strings at one point and the halves are spliced to create new string. The mutation operator creates new string by making changes in a single string. In each generation, a new population is created by taking advantage of the fittest strings of the previous generation. After a fixed number of generations or until a termination condition is satisfied, the algorithm converges to the best string, which hopefully represents an optimal or suboptimal solution to the problem. The following List 4-3 description presents a high level pseudo-code of a typical GA for clustering.

```
GAs-based Clustering Algorithm (K, n, g, P, C)
Where:
K - Number of Clusters
n - Population Size
g – Generation
P(g) - a population of strings
C(g) - new population of strings
g = 0;
int[n][k] P(g); // ∀string∈P(g)[i], i=1…n
Initialise P(g);
Evaluate P(g);
WHILE (no termination condition is satisfied) DO {
   Associate a fitness value to string;
    Recombine P(g) to yield C(g);
    Evaluate C(g);
    Select P(g+1) from P(g) and C(g);
    g = g + 1;
}
 RETURN the best generation of strings;
```

List 4-3. GAs-based Clustering Algorithm

As to the MDG partitioning problem, the GAs-based clustering is mostly differ in the meaning of strings in the population, fitness function, initial population and evolutionary operators. The goal of MDG clustering analysis is to find a “good” partition from the MDG, so that closely related modules/components (*genes*) are grouped into subsystems (*chromosomes*). GAs-based clustering analysis treats such task as an optimisation problem. By implementing the fitness function and definitions that are designed specifically for the MDG partitioning environment, GAs-based clustering analysis can express excellent execution efficiency on the large-sized software system. Generally, 5 main steps are included in the process of GAs-based MDG partitioning:

Step 1. Define specific fitness function to measure candidate solutions.

The fitness function is used to quantify the quality of an MDG partition so that a subsystem with a high fitness represents a better solution to the problem than a subsystem with a lower fitness. The MDG-based fitness function needs the data of inter and intra-connectivity. They are defined as below:

Intra-connectivity is a measure of the density of connections between the modules within a single subsystem. It is calculated by the following formula:

$$A_i = \frac{\mu_i}{N_i^2}$$

A_i represents the intra-connectivity of subsystem i which has N_i modules and μ_i intra-edge dependencies (relationships to and from modules within the same subsystem). A_i is the number of intra-edge dependencies divided by the maximum number of possible dependencies between the modules of subsystem i, which is N_i^2 .

Inter-connectivity is a measure of the connectivity between distinct subsystems, represented as:

$$E_{i,j} = \begin{cases} 0 & i = j \\ \dfrac{\varepsilon_{ij}}{2N_iN_j} & i \neq j \end{cases}$$

$E_{i,j}$ represents the inter-connectivity between subsystem i and j, each consisting of N_i and N_j modules respectively with ε_{ij} inter-edge dependencies (relationships between the modules of both subsystems). $E_{i,j}$ is 0 when there are no inter-dependencies between subsystem i and j; and 1 when every module in subsystem i depends on every module in subsystem j and vice versa.

The MDG-based fitness function is defined as *Modularisation Quality* (*MQ*). Given a k partitioned MDG, *MQ* can be calculated as:

$$MQ = \begin{cases} \dfrac{\sum_{i=1}^{k} A_i}{k} - \dfrac{\sum_{i,j=1}^{k} E_{i,j}}{\frac{k(k-1)}{2}} & k > 1 \\ A_i & k = 1 \end{cases}$$

MQ is obtained by subtracting the average inter-connectivity from the average intra-connectivity. Thus the *MQ* is bound between -1 (no cohesion within subsystems) and 1 (no coupling between subsystems). *MQ* can be used as a measurement of the fitness of candidate solution. Therefore, the *MQ* of the newer generation is larger then the *MQ* of the original generation. Through establishing a tradeoff between

inter-connectivity and intra-connectivity, *MQ* rewards the creation of highly cohesive subsystems and penalises the creation of too many inter-cluster dependencies.

Step 2. Select encoding scheme.

Encoding the membership of modules in the solution is a key issue when using GAs. For many problems in the industrial engineering world, modules can be encoded in a real-number representation. The position of a gene in chromosome is used to represent a module, and the value of the gene is used to represent a subsystem in which the corresponding module is put. For example, the chromosome *S* (*S*= 2 3 1 2 1) encodes a solution where the first module is in subsystem 2, the second in subsystem 3, the third in subsystem 1, the fourth in subsystem 2 and the fifth in subsystem 1. The representation of the arrangement problem is illustrated in Figure 4-10.

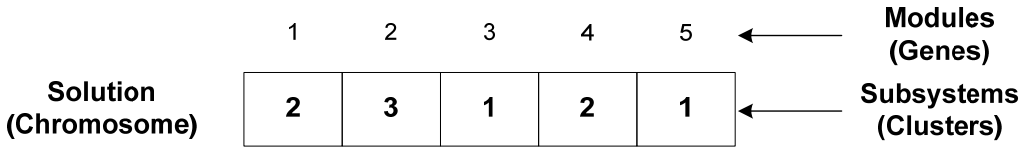

Figure 4-10. Representation of Membership of Modules

As shown in Figure 4-11, this encoding scheme reflects in MDGs, where each node is assigned to a unique numerical identifier (e.g. the first node is assigned the unique identifier N1, the second node is assigned the unique identifier N2, and so on). These unique identifiers define which position in the encoded string will be used to define the cluster of that node. Therefore, the first character in string *S*, which is 2, indicates that the first node (N1) is contained in the cluster labeled 2. Likewise, the second node (N2) is also contained in the cluster labeled 3, and so on. Finally, an encoding on a string *S* will be defined as $S=s_1\ s_2\ \ldots\ s_i\ \ldots\ s_N$, where N is the number of nodes in the graph and s_i ($1 \leqslant i \leqslant N$), identifies the cluster that contains the i^{th} node of the graph.

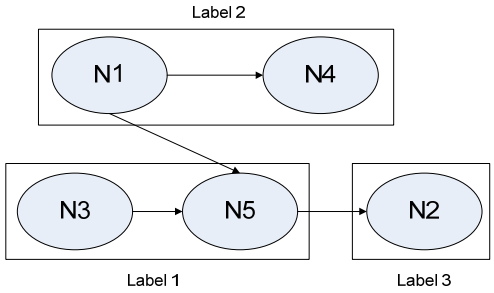

Figure 4-11. A Sample Partition

This representation has the advantage that the chromosomes are of constant length (equal to the number of modules), which allows for the application of standard genetic operators.

Step 3. Create initial population.

Once the fitness function and the encoding scheme have been properly defined, the GA starts by generating an initial population of partitioning solutions. Ideally, the first population should offer a wide diversity of genetic materials and the solution pool should be as large as possible so that any solution of the search space can be engendered. However, the size of the population raises few problems too. The larger the population is, the easier it is to explore the search space, but a large population requires much more computational cost, memory and time.

Initial population is very important for efficiency of GA. If the initial population is created in a way that its chromosomes have better talent for reaching an efficient answer, GA could converge quickly. According to the previous legacy system understanding, a good initial population can be obtained quickly by combining domain knowledge. Using domain knowledge to hybridise a GA would improves its performance. Thus, the mean fitness of the population is already high and it may help GA to find good solutions faster. But for doing this one should be sure that the solution pool is still large enough. Otherwise, if the population badly lacks diversity, GA will just explore a small part of the search space and never find global optimal solutions.

Step 4. Execute genetic operators.

The process of executing genetic operators is the heart of GAs. In this step, the GAs-based clustering creates new and hopefully fitter subsystems through an operating cycle which consists of four phases:

a) Selecting parent solutions.

b) Crossing the parent solutions to create children.

c) Mutating obtained solutions to recovering the lost genetic materials.

d) Replacing old solutions in the population with the new ones.

Selection is an operator that picks parent solutions out of the population for further manipulations. There are many existing selecting methods in GAs. Since the initial population has limited the amount of the candidate solutions, the convergence rate of the GA will much quicker and easier compare to the all possibilities covered GAs. Therefore, a *tournament selection strategy* is applied to choose parental members for reproducing population. The best solution from the tournament is the one with the highest fitness. Tournament competitions and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding solutions. After the selection process, the new population is enriched with better solutions.

Crossover operator is utilised to generate offspring from the mating poll. In MDG representation, modules that encode dependant characteristics in a solution should be passed together to the offspring. Consequently, a uniform crossover is implemented as recombination operator to avoid the modules position problem caused by N-point crossover. Each module in the offspring is created by copying the corresponding module from one or the other parent chosen according to a random generated binary crossover mask of the same length as the solutions. Where there is a 1 in the crossover mask, the module is copied from the first parent, and where there is a 0 in the mask the module is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offsprings, therefore contain a mixture of genes from each

parent as illustrate in Table 4-3.

Parent 1	3 4 3 1 4 2 2 1
Parent 2	2 4 2 1 3 1 3 4
Mask	1 1 0 1 0 1 0 0
Child 1	3 4 2 1 3 2 3 4
Child 2	2 4 3 1 4 1 2 1

Table 4-3. Uniform Crossover

Mutation implements kind of hill-climbing mutation operators that do mutation only if it improves the quality of the solution. In such operator, a parent is considered and a mutation chromosome is randomly generated. For a 1 in mutation chromosome, the corresponding module in parent solution is flipped and child chromosome is produced.

Replacement is the last phase of the operating cycle. Two parents are drawn from a fixed size population and breed two children, but not all four can return to the population. *MQ* is therefore utilised to determine which of the current members of the population, if any, should be replaced by the new solutions.

Step 5. Terminate clustering.

The GAs-based clustering stops when either of the two conditions is met:

a) The genetic operators have evolved specified number of generations.

b) There is no change to the best fitness of population.

The termination or convergence criterion finally brings the search to a halt and display the best solution found. This result maybe sub-optimal, but is often adequate.

Therefore, components/modules are partitioned into corresponding clusters on the basis of the best solution. That is to say, legacy system is divided into different

subsystems.

4.3.4 Fine-grained Characteristics

The GAs-based clustering partitions legacy systems into high cohesive and loosely coupled subsystems. The term *loosely coupled* is often used as a synonym for *coarse-grained*. However, the obtained subsystems in this chapter are still regarded as *fine-grained*. Because the term loosely coupled in this chapter only represents the degree of coupling between legacy functionalities. Compare with the term in SOA, the coarse-grained character should be examined from following aspects:

- Interface coupling
- Data (type systems and message formats)
- Version resiliency
- Transport independence
- Expected interaction patterns
- Conversations
- Ability to mediate
- Dynamicity

The fine-grained subsystems contain valuable business logic, rules, information, workflow and sometimes a user interface that the users have grown accustomed to and probably like. While such self-contained character helps subsystems gain advantages in efficiency and independency; there still have some distinct drawbacks that prevent them evolving into SOA.

Legacy subsystems are regarded as single blocks that contain integrated part of domain-specific business knowledge. Most of the legacy subsystems require data passed to them through their specific interfaces. In fact, conversion of a monolithic legacy subsystem to reusable services can be performed with an analysis that takes into account the existing business knowledge, system interface, workflow, system processes, functional logic and application data. The idea is to break up the monolith and harvest reusable legacy assets. These legacy assets can be reorganised into service layers by

leveraging portals, process automation and shared business knowledge store infrastructures. System interfaces will be replaced with service interfaces to make them adapted to SOA.

4.4 Summary

This chapter discussed the cognitive aspect of large-scale legacy system comprehension. The comprehension process can be divided into two parts: Architecture Recovery and Architecture Partitioning.

Differ from the traditional legacy system understanding strategies which emphasis on the internal behavior of legacy systems and recover the high-level system architecture from low-level source code, the proposed approach focused on the external behavior of legacy system. The high-level system understanding that advocating here uses a combination of top-down and iterative refinement strategies to acquire knowledge about a software artifact through analysis, abstraction and generalisation. This is a deductive process begin with a preexisting notion of the functionality of the system and earmark individual components of the system responsible for specific tasks. And then, create, verify and modify hypotheses according to the key words in the source documents until the entire system is explained by a consistent set of hypotheses. Finally, the legacy system architecture can be recovered by matching the patterns to the specific domain.

Once retrieve the legacy system architecture, the next step moves to the architecture partitioning problem. There are many ways to partition a software system, but the most suitable method should make use of the previous obtained domain and architecture knowledge to achieve an efficient and accurate partitioning. Therefore, a GAs-based clustering is implemented which treats system partitioning as an optimisation problem and uses a Genetic Algorithm (GA) to search the extraordinarily large solution space of all possible MDG partitions. The advantage of this approach is its outstanding efficiency when applied on large-sized MDG.

Generally, the two phases introduced in this chapter build a solid foundation for the service-oriented transformation operation in the following chapters.

Chapter 5
Business Rules Extraction

Objectives

- The need for extracting business rules and what are they look like.
- Extracting business rules by using data-flow and Information-flow analysis.

Class diagram is a general description that used as initial source of information by reverse engineers in recovering legacy system specification. One of the problems with legacy systems is their lack in coping with the changing business environments. In this chapter, a solution is introduced to build coarse-grained volatile objects instead of fine-grained persistent objects.

This chapter introduces a data-centred approach provides an integrated model with representations, algorithms and tools that can be used to automatically analyse legacy code and answer important queries related to software maintenance.

5.1 Business Process Model Abstraction

The structures obtained by using top-down strategy represent explicit call and control-flow graphs of legacy systems. Such graphs are very helpful in understanding how systems handling business. From the business operating perspective, each subsystem is responsible for certain scope of duties that corresponding to their specific domain knowledge.

Business makes sensible decisions by invoking subsystems to apply rules to facts based on their embedded domain knowledge. These rules are known as business rules.

When business people and information systems access to appropriate subsystems at the opportune moment, they can make informed choices in a timely manner. Comparing with the Figure4-7, legacy system architecture can be abstracted as the following business process model as shown in Figure 5-1:

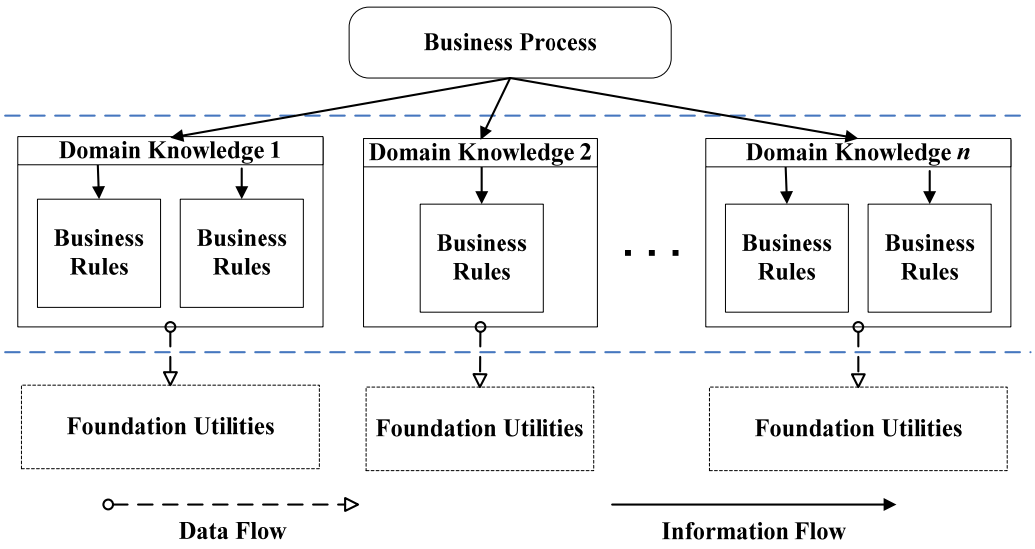

Figure 5-1. Business Process Model

In this process architecture:

- Business Process is the rounded rectangle that spans the entire system and can make use of the underlying services.
- Domain Knowledge exposes high-level, composite business logic to the system. The functions and information match closely to the semantics and syntax required of business processes. Domain knowledge is labelled with the names of subsystems.
- Business Rules provide business-related services that are specific to a business knowledge domain but may not be exposed outside of the domain.
- Foundation Utilities locate on the lowest tier to provide simple functionalities.
- Information Flows connect process steps, showing and enforcing specific operations to the processes.

- Data Flows show how processes produce and/or consume specific data.

Business process model provides a wonderful abstraction for building business systems, but all too often used in the past. Rather than contributing to an overall flexible, agile enterprise, it is more suitable to build siloed applications. In 0, specific domain knowledge is fixed in each subsystem. When business practice or policies change, as they often do, it is difficult to correctly reflect those changes in the application implementation. For the purpose of adapting business agility, the domain knowledge embedded in these prototype subsystems should be extracted and implemented as separate resource. This resource represents the abstraction of business logic and can serve as a guidance service that influences the collective behaviour of information systems.

The principle of abstraction is normally used in the context of ensuring that a service is independent of a specific implementation and other detail. In terms of enabling agility, the abstraction is used to:

- Remove implementation specific references from legacy subsystems.
- Hide data or behaviour in the implementation that is specific only to the internal working of business knowledge and not important to the service consumer.
- Transform data types of business rules that are specific to the legacy system into a standard format.

Some of these can be seen as similar to principle of encapsulation in object and component technology. However, the goal is not just to hide details behind an interface, but also extract business rules away from the implementation technology and any other implementation specific features to reduce coupling. Therefore, when building a new SOA, this abstraction can be built into the services from scratch.

5.2 Defining Business Rules

Before executing business rules discovery, it is better to start with a description of

what to look for rather than how to find them. At present, there is no industry standard definition for the term business rule. This thesis adopts a reasonable definition put forth by the Business Rules Group (2000) which states a business rule is: “A statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business.” Although reasonable, this is a fairly broad definition. It is necessary to come up with a more restrictive definition of business rules that applies to the work of business rules discovery.

5.2.1 Abstraction Level

An issue that is sometimes confused with atomicity of business rules is their level of abstraction. Although subsystem has dramatically reduced the workload of extracting rules from legacy codes, the textual representation of subsystem does not provide much guidance in the business rules discovery. Therefore, clarifying the levels of abstraction of business rules and rule dependencies becomes an important step.

A generally used high-level abstraction of business is decision tree. A decision tree is a classifier expressed as a recursive partition of the instance space. A decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called root that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called internal node or test nodes. All other nodes are terminal nodes and called leaves.

Many problems, including business rules, can be expressed at different positions in a decision tree. The highest levels (i.e. the root node of a decision tree) are very useful for understanding the concepts that underlie particular domain knowledge. However, they are often not concrete enough to be implemented in information systems as they stand. For example, a credit card company can have rules like “We will not tolerate fraud.” or “Our customers are our first priority”. As statements they are fine, but they are nowhere nearly atomic enough to be recognised as business rules. Nevertheless, as reverse engineers trace the decision tree from root to leaves, rules become more and more specific and a point is reached where they can be implemented.

Obviously, subsystem is not a correct level to start business rules extraction because they are divided by domain knowledge. From a decision tree point of view, domain knowledge is like some main branches, which are too general to use as business rules. Contrary, the modules of subsystems are usually response for more concrete tasks and can be looked as leaves or terminal nodes in a decision tree. Therefore, modules should be the suitable candidates for extracting business rules.

5.2.2 Business Rule Templates

After clarifying the business rules abstraction level, the next step is then moves to a more concrete work of describing business rules. For business rules to be used effectively, a wide range of people must be able to read and write business rules without support from technical expertise. The best way to achieve this target is expressing the original business rule statements in a colloquial natural language. However, according to the business rules definition, there are different types of business statements that qualify as business rules. The words used in unconstrained natural language to describe business rules are arbitrary. To deal with these difficulties, the most convenient way is defining a suitable set of business statement patterns to be used as business rule templates. A set of patterns allow business rules to be represented as reasonably natural-sounding business sentences without introducing all the difficulties associated with full natural language processing. Besides saving valuable analysis time, rule templates can also help to enforce the consistency of business rules.

The rule templates introduced in this section are in the form of statements which are all elaborate versions of one basic form:

```
<term1> <operator> <term2>
```

In this simple rule template, business rule is expressed as a combination of clauses where “`term`” is a noun or noun phrase with an agreed upon definition and “`operator`” is any operator that makes sense for the particular term type. For example, in business statement “Customer credit rating code is A.”, there are two terms (“customer credit rating code”, “A”) and one operator (“is”).

Business rules are related with different aspect of business, In order to build precise business rules, there should have specific business rules templates for each kind of business. However, classifying business rules is a sophisticated work. At present, many individuals have proposed various business rule classification schemes and there is no one universally adapted. The usefulness of a business rule classification scheme depends on the purposes served by it for the intended object. For the purpose of efficiently implementing business rules, a classification scheme that classifies each business rule according to what it does is most intuitive to business professionals. Based on this scheme, there are four different kinds of business rule needing templates which are: terms, facts, derivations and constraints. The details for all kinds of business rules that might appear in each business rule template are list as below:

Term:

A term is an application of a single definition to a word or phrase. It turns out to be noun or noun phase. In business rules, term can define any one of the following:

- Entity
- Attribute
- Value
- Value set

For example, “customer” can be defined as an entity and “customer credit rating code” is an attribute of the “customer” entity. “male” is a constant value. “Mon, Tue, Wed, Thu, Fri, Sat, Sun” is a set of values that may become a domain. The template of term is:

Template 1: `<term> IS DEFINED AS <text>`

Fact:

A fact is a statement that represents relationships among entities or the association of an attribute to an entity. It turns out to be prepositions and verb phrases which connect terms into sensible, business-relevant observations Fact includes:

- Entity to entity relationship
- Entity to attribute assignment
- Super-types and sub-types

For example, “Customer can place order.” is a fact that is a relationship between “customer” entity and “order” entity. “Customer qualifies for customer credit rating code” may materialise as a m:n relationship between “customer” entity and “customer credit rating code” entity and will then become an association of the attribute “customer credit rating code” with “customer” entity. The templates for expressing fact are:

Template 2: `<term1> IS A <term2>`

Template 3: `<term1> <verb> <term2>`

Template 4: `<term1> IS COMPOSED OF <term2>`

Template 5: `<term1> IS A ROLE PLAYED BY <term2>`

Template 6: `<term1> HAS A PROPERTY OF <term2>`

Derivation:

A derivation is a declarative statement that applies logic or computation to discover new information. Derivation turns out to be executable logic or algorithm that result either in a new knowledge of data. Therefore, derivation has two types:

- Inference
- Mathematical calculation

Inference derivation statements derive new knowledge from existing knowledge. An inference rule example is: “If an order amount exceeds £1000, then the customer’s qualifies for a 10 percent discount.” This rule tests conditions (“if order exceeds £1000”) and upon finding it true, establishes the truth of a new fact (“qualify for 10% discount”). Computation derivation statements refer to a result of computation on numeric data for arriving at a value of a term. For example, “Total order amount is computed as the sum of the line item amount(s)”. The new piece of information is “total order amount”, which is derived from “line item amount(s)” by using addition algorithm. The templates

for expressing derivation rules are:

Template 7: `<term1> IS COMPUTED AS <formula>`

Template 8: `IF <term1> <operator> <term2, value, value list> THEN <term3> <operator> <term4>`

Constraint:

A constraint is a mandatory or suggested restriction on the behavior of the business event. Constraint turns out to be executable logic that uses information as input and action as output. The type of constraint includes:

- Mandatory constraint
- Suggested constraint
- Action enabler

Mandatory constraint is a complete statement that expresses an unconditional circumstance that must be true or not true for the business event to complete. For example, “A customer must not have more than 10 open orders at one time.” The input information in this constraint example is the number of customer order and the output action is allow or deny the new order. A suggested constraint does not force the circumstance to be true or not true, but merely warns about it. Because a suggestion only warns and does not reject, it provides a freedom of choice. For example, “A customer should not have more than 10 open orders at one time.” In this example, suggested constraint rule highlight the customers who have more than 10 open orders and allows the human to make the decision. Another constraint circumstance is testing conditions and upon finding them true, initiates an action. For example: “If a customer is high risk, then notify the customer services manager.” The templates for constraint rules are:

Template 9: `<term1> MUST (SHOULD) (NOT) HAVE <at least, at most, exactly n of> <term2>`

Template 10: `<term1> MUST (SHOULD) (NOT) BE <comparison n> <term2>, <value>, <value list>`

Template 11: `IF <rule phrase(s)> THEN <constraint of any of the above types>`

Template 12: `IF <term1> <operator> <term2> THEN <action>`

Business rule templates provide a guideline to develop business rules. Base on this theory, a business rule is an aggregation of information and structural business rule components. Business rule templates also reveal a theory that business process can be viewed as a sequence of business rules that define how the thread of control is passed from one activity to another and in what circumstances the transition can happen. Business rules extraction work is then simplified as identifying the symbols that accord with business rules templates components in information process. With these symbols as clues software engineers can easily pick up whole business rules according to the business rule templates.

5.2.3 Validation Standard

The main problem from rule templates approach is that it tends to blur the distinction between specification and realisation. In some cases, this results in the meaning of event being bent toward a situation governed by technology (i.e. new message enters queue), rather than business (i.e. customer application received). In order to avoid missing anything by the realisation issues, the business rule definition take here is therefore add the features of business rules to the ECA approach as supplement. Below is a simple list of the characteristics that software engineers would like to see in business rule statements:

- Atomic: can not be broken down any further without losing information;
- Unambiguous: have only one, obvious, interpretation;
- Compact: typically, a single short sentence;
- Consistent: together, they provide a unified and coherent description;
- Compatible: use the same terms as the rest of the business model.

These universal characteristics apply across any language, tool, application domain or any other sort of division. Rules that don not meet these standards have to be

considered and broken down into detailed (structural) rules.

The business rule features discussed in this section can be utilised as a quality control during the business rules extraction process. Only the business rules that fit with the previous agenda could possibly be approved and implemented in the target systems.

5.3 Business Rule Extraction

From the previous discussion, software engineers have got a clear idea of what the desired business rules should look like. Now it is the time to start finding them.

The raw legacy subsystems passed to this stage are highly complex and mix together code that is concerned with application logic, data and user interface. In order to separate “pure” business rules, reverse engineers first have to decide from which resource the business rules should be extracted. Experience in reengineering shows the documents of legacy systems are usually inconsistent with code. Therefore, source code becomes more reliable resource than any documentation to get business rules. Based on this understanding, the major work of this section will concentrate on the code-level analysis.

5.3.1 Domain Variables Identification

Legacy systems usually involve much more data movement and manipulation activities than sequencing, branching or looping activity, in other words, they tend to be data-rich. Business rules in such systems are directed toward creating, maintaining and utilising data. Since data reflect the execution of business rules, it becomes an ideal starting point to locate potential business rules from the enormous legacy source code.

The identification of (Variables Of Interests) VOI is not a trivial task within large and complex legacy system. Program represents data by using variables. A typical large system usually contains thousands of even hundreds of thousands of variables. However, only a subset of them is related with business rules. Those useful variables are called domain variables because they are mapped to important objects and expose domain concepts in applications. Domain variables can be of two types:

- Pure Domain Variables: those that can be directly mapped to objects in the application domain.

- Derived Domain Variables: those that dependent on domain variables only. But if the variable is a constant, it is classified as a pure domain variable.

Given a variable-of-interest, reverse engineers can easily find all the relevant statements and which goes into definition of business rules. In order to locate business rules, reverse engineers must initially focus on domain variables. For such a requirement, it is wise to take advantage of the result from Section 4.3 in which legacy systems have been divided into logically coherent subsystems according to their specific domain knowledge. The modules that construct these subsystems are regarded as black boxes and map their inputs to outputs. Obviously, the input and output variables of each module must contain some of the pure domain variables. The left work is identifying the other pure and derived domain variables in the modules.

Domain variables identification can be accomplished by investigating the information-flow of modules. This method is derived from the information-flow algorithms which have been introduced in Chapter 2. For any program modules there have three kinds of information-flow relations:

Relation 1: A relation between the values of input variables on the module entry and the values of the module expression parts.

Relation 2: A relation between the values of the module expression parts and the values of output variables on the module exit.

Relation 3: A direct relation that associates the values of input variables on entry to the module with the values of output variables on exit from the module.

These relations are corresponding to the information-flow algorithms respectively as below:

$$Relation\ 1 \rightarrow \lambda_S\ (from\ v_i\ to\ e)$$
$$Relation\ 2 \rightarrow \mu_S\ (from\ e\ to\ v_o)$$
$$Relation\ 3 \rightarrow \rho_S\ (from\ v_i\ to\ v_o)$$

These relations enable reverse engineers to identify the program statements which may play a role in the transmission of information, either from or to particular program variables. For the purpose of identifying domain variables, comparing with λ_S and μ_S algorithms which are both represent the relations between expression and variables on the statement level, ρ_S algorithm can do a better job because it focuses on the variable level. ρ_S algorithm indicates those members of v_i whose values on entry to statement S may be used in obtaining the value of each member of v_o on exit from S. Thus, a tabulation of ρ_S must include every feasible execution paths from input to output variables within statement S. Through tracing the effect-relation between variables, ρ_S relation can then find out all pure and derived domain variables of S.

Leaving the theory of ρ algorithm unchanged, it is easily possible to extend the formula in Chapter 2, which is originally applied to program statements into program modules. The old formula for calculating "$v_i \rho v_o$" relation is then changed as:

$$\rho_M = \lambda_M \mu_M \cup \prod_M \quad where \begin{cases} M \rightarrow \text{candidate module} \\ \lambda_M = \lambda_S \\ \mu_M = \mu_S \\ \prod_M = \{(v,v) \in V \times V \mid v \in P_M\} \end{cases}$$

Above formular signifies *"The relations between input and output variables of a module M include either: (1) the value of input variables may be used in obtaining the value of some expressions in M, which in turn may be used in obtaining the value of output variables, and (2) the Cartesian product of preserved variables (P_M) if the input variables been outputted by M without any changes."*

In order to obtain the tabulation of ρ_M relations, reverse engineers first have to divide the set of all variables V into two groups:

a) The set of variables which M contains any assignment to is in an aggregation called D_M

b) Those variables which M does not traverse any assignments to are in another group called P_M.

Obtaining D_M and P_M is the precondition of calculating λ_M and μ_M. In practical situation, reverse engineers are usually had to deal with various programs which have different parse tree applied on them. No matter what kind of programs used, the approaches of calculating their information-flow relations are consistent. The key is looking the large-sized program module as an aggregation of program statements. By obtaining information-flow relation for each statement of *M* in turns in the course of a post-order traversal of the module, reverse engineers can easily get the whole information-flow relations of *M*. On visiting a tree node which represents an assignment statement, reverse engineers obtain its information-flow relations directly from the formulas; on visiting a node which represents a compound, conditional, or repetitive statement S, reverse engineers use the appropriate formulas (Table 2-1) to derive the information-flow relations for *M* from those of its constituent statements. (These are immediate successors of *M* in the parse tree, which will already have been visited in the post-order traversal.) In this way, at each stage it is only necessary to retain the information-flow relations for the immediate successors of the tree nodes on the path from the tree root to the node currently being visited. Finally, with the λ_M and μ_M relations obtained, the ρ_M relation can be easily figured out.

For ease of explanation, the process of domain variables identification is demonstrated by a simple discount calculator module in the form of C program, as depict in List 5-1.

```
1  calculator(total)
2  {
       Base=1000
3      if(total>=10*base)
4          indicator=10;
5      else
6          indicator=total/base;
7      switch(indicator)
8      {
9          case 10: discount+=5;
10         case 9:
11         case 8: discount+=2;
12         case 7:
13         case 6:
14         case 5:
15         case 4: discount+=3;
16         case 3:
17         case 2: discount+=3;
18         case 1: discount+=2;
19         case 0:
```

```
20     }
21     price=total*(1-discount/100)
22     return (price);
23 }
```

List 5-1. Delivery Cost Calculator Module Sample

The variables that occur in this module include: *total*, *base*, *indicator* and *price*. Among them, *total* and *price* are input and output variables of the module. According to the definition of D_M and P_M, by checking all assignment statements in this module, the set of all variables can be divided as below:

$$D_M = \{indicator, discount, price\};\ \ P_M = \{total, base\}$$

The λ_M and μ_M relations for the module are given in Figure 5-2 and Figure 5-3 (here each relation is represented by its Boolean relation matrix).

λ_M

	3	4	6	7	9	11	15	17	18	21
total	0	0	1	1	0	0	0	0	0	1
base	0	0	1	1	0	0	0	0	0	0
indicator	0	0	0	0	0	0	0	0	0	0
discount	0	0	0	0	0	0	0	0	0	1
price	0	0	0	0	0	0	0	0	0	0

Figure 5-2. λ_M Relation of Sample Module

$\mu_M =$

	total	*base*	*indicator*	*discount*	*price*
3	0	0	0	0	0
4	0	0	0	0	0
6	0	0	1	0	0
7	0	0	0	0	0
9	0	0	0	1	1
11	0	0	0	1	1
15	0	0	0	1	1
17	0	0	0	1	1
18	0	0	0	1	1
21	0	0	0	0	1

Figure 5-3. μ_M Relation of Sample Module

As a result the ρ_M relation of the sample module can be computed:

$$\rho_M = \begin{array}{c} \\ total \\ base \\ indicator \\ discount \\ price \end{array} \begin{array}{c} \begin{array}{ccccc} total & base & indicator & discount & price \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Figure 5-4. ρ_M Relation of Sample Module

Since the input variables `total` and `base` are chosen as the initial domain variables, the backward trace ρ_M in information relations algorithm should be used to identify all domain variables which are affected by input variables in current module. Therefore, reverse engineers can inspect the pair (`total`, `*`) and (`base`, `*`) in ρ_M, includes: (`total`, `total`), (`total`, `indicator`), (`total`, `price`), (`base`, `base`) and (`base`, `indicator`). Based on this discovery, reverse engineers can further inspect all variables affected by input variables: `total`, `base`, `indicator`.

Similarly, as the output variable `price` has been chosen, the forward trace ρ_M should be applied to identify all domain variables which will have effect on output variables. The result is: `total`, `discount`, `price`.

Finally, put these two sets of variables together, reverse engineers can get all the domain variables of the sample module (`total`, `base`, `indicator`, `discount`, `price`).

5.3.2 Relevant Program Statements Extraction

Locating business rules in large complex legacy systems is always a challenge to every reverse engineer. A widely used solution is split the program into smaller units to assist program comprehension and business rules extraction. The aim is retrieving all the program statements that are relevant to a particular computation while the irrelevant parts be deleted. However, most practical implementations failed to meet this target

because business rules are often found tightly embedded in legacy systems. For this situation, utilising program slicing tools is a quick solution but it usually leads to changes on legacy system semantics. Contrarily, a manual comprehension can increase the quality of the extracted business rules but it is too costly and also lost the original idea of reducing the size of legacy systems.

With the domain variables identified, it is possible to quickly carve out the minimal subset of program statements without affecting the embedded domain knowledge. Typically, program slices are extracted from the legacy code up to certain point of interests, referred to as business rules. In such a way the original semantics of the legacy program can be conserved.

Program statements extraction aims at preserving code behaviour on any input value. The criterion of judging the consistency of domain knowledge implies that the original and the retrieved program compute the same result for all domain variables. To meet this requirement, reverse engineers must retrieve all of the expressions in *M* whose values have direct or indirect impact in obtaining the value of v_o. This notion of expressions appears to be very similar to that of the domain variables. By joint studying, one can discover the expressions which manipulate these domain variables can satisfy the program retrieving conditions. Therefore, let *v* be any domain variable of *M*. For the set of retrieved expressions E_M^v , there have:

$$E_M^v = \{ e \in E \mid e \mu_M v \}$$

Where *E* denotes the full set of expressions in *M* and *e* denotes any retrieved expression from *M*. As a result, E_M^v represents the set of expressions in *M* whose values may be used in obtaining the value of v_o on exit from *M*.

Since E_M^v contains all of the relevant expressions to v_o, reverse engineers can safely replace every statement within *M* which does not contain any member of E_M^v with an empty statement. The obtained partial program is equivalent to *M* in the sense that, for any set of input values, the values of v_o on exit from *M* and E_M^v are identical. Thus, all the program statements that relate with domain variables can be retrieved.

Recall the calculator module discussed in last section. The module only outputs one variable "`price`", then all the program statements that associate with `price` can be retrieved.

From this very small example, it is obvious that the extraction of partial statements can be useful as an aid to understanding how the exported values of particular program variables are obtained.

The above statements extracting process presupposes reverse engineers have understood the transaction process of module, only in terms of input and output data. The aim is to take only those statements which yield the output data, both directly and indirectly, starting from the given input data. These statements, unlike conventional program slicing, do not include either the statements necessary to get input data or the branch and loop conditions which are used to control the activation of the function. In addition to the input/output parameters of the function, the slicing criterion depends on an initial statement. This statement, which is usually the last instruction of the function to be recovered, is difficult to identify because it requires reading a lot of code. The proposed approach overcomes this problem by providing a scavenging algorithm which invokes transform slicing but does not depend on statement numbers. A set of candidate functions are produced and evaluated with respect to their expected behaviour. Although this concept validation step is not automatic, the limited size of candidate functions makes this task easier than looking for the last function instruction in the original source code.

5.3.3 Business Rules Generation

The key step of generating business rules is to get the business items that are actually implemented in the current slice. Then the business items extracted from every slice should be integrated as a whole. As stated in Section 5.2.2, all business rules are generated based on business rule templates. These rule templates are looked as an appropriate starting point for business rule refinement process at the business process activity level. The business rules that underline the business activities are first described

in a natural language. In subsequent steps, these rules are refined in a structured way as a set of structured rules representing the business process at different abstraction levels.

A set of business rules will apply to a business action. Business actions are common practices that regularly occur in a specific industry. Examples of business actions for the Retail industry might include Computing a Discount, Refund Customer and Restock Inventory. If select the business action Calculate Discount, many business rules could then be defined to support it. Examples of applicable business rules may include:

- Orders greater than 500 units receive a 30% discount.
- Orders greater than 100 units receive a 15% discount for preferred customers.
- Orders greater than 100 units receive a 10% discount.

5.4 Summary

Modeling business rules as separate entities offers great flexibility. Especially in the e-commerce domain, this can be a valuable advantage, since the business analyst, who ideally authors the business rules, does not need to have programming knowledge to change the rules. Typically, changing the business rules happens more often than changing the large e-commerce applications. One of the most important facts about business rules is that they are declarative statements, they specify *what* has to be done and not *how*.

Business rules are constraints on business. It describes what business must do, or what they cannot do. Decoupling business rules from the application can provide a number of advantages. Because business rules are external to the applications that depend upon them, the variable business logic contained in them can easily be changed. This can be made by a rule management.

From the business rule characters, it is obvious that business rules are usually deal with data (either input or output data); therefore, a data-centered approach would be the best practice to discover business rules from complex legacy codes.

For ease of understanding, the initial business rules are usually written in plain language, represented by using restricted natural language sentences.

To extract business rules from legacy systems. By implementing data-flow and information-flow analysis, business rules can be automatically extracted. This is very important for big scale legacy information systems.

Variables that used by these subsystems are then called as domain variables. Domain variables are mapped to important objects in subsystems, so the analysing work will initially focus on domain variables to extract business rules.

Thanks to the partitioned legacy subsystems, reverse engineers can quickly locate interest domain variables within limited application domains instead of searching them from hundreds or thousands of variables that often occurs in data-rich software systems.

Reverse engineers have to keep in mind that the ECA structured business rules only provide information to human. They have to be translated into other machine understandable representations. The combination of business rules and XML provides adaptivity in information systems.

Chapter 6

Representing Business Rules in XML

Objectives

- To illustrate the XML-based business rules.
- To define the transformation between business rules and XML.

Although ECA structure restricts the natural language sentences used in expressing business rules. Software engineers still need to keep in mind that this is only a high level representation and they must bridge the gap between the operational level at which the programs are and the conceptual level of the business knowledge located.

The XML formatted business rules are not structured around the development of a single application. Instead, they look at many of the pieces that might be used to build BRM subsystem based on the SOA. One of the most important advantages is that they provide better interoperability between rule engines and reduce the time required to learn how to use a new rule engine. Software engineers should be able to change rule engines without modifying business rules.

For the purpose of consistent rule representation and implement, the natural language business rules have to be transformed into a machine understandable format. This chapter introduces an approach that transform the conceptual natural language business rule to operational XML documents which preserve semantic business rules.

6.1 Advantages of XML

The target audience of ECA structured business rules is either software engineers or

business professionals. However, it is difficult for computers to decipher the contents of these documents because they are written in natural language. There is a critical need for devising rules or guidelines for converting ECA structured business rules into a machine understandable format.

At present, a number of projects exist with the aim of defining a standard business rules language. Representing rules as XML makes sense for several reasons:

Interoperable: Business rules engines are the main consumers of business rules. Currently, there exist many rules engines in the market. Most of them use business rules in proprietary format and no mapping systems exist for any pairs of them. Information systems are often required to integrate rules engines from different vendors to gain more competitive business. When engineers change the used or add new rules engines, they have to spend much energy on rewriting business rules. In order to let various business rules engines share the same rules source, the desired business rules language should independent to any of the rules engines. XML is just such a language that uses the common core of business rules concepts to develop a neutral, flexible representation. Furthermore, XML can easily be translated into the native format supported by a specific rule engine as needed.

Editable: It is necessary to put together a rule editor which has a graphical interface to allow nontechnical people to add and modify business rules by pointing and clicking instead of programming. Some rule engines come with an integrated rule editor for developing business rule. But these editors only let users create rules in the native language of the rule engine, not in vendor-independent XML. If users want to keep interoperability, they have to develop a specific XML rule editor. The good news is that XML can make this development fairly painless. Business professionals that work with XML need not to worry about parsing existing rules or writing out completed rules, because they can use existing XML-parsing and -writing libraries to do the hard work.

Searchable: Many businesses repository have a collection of numbered or otherwise indexed business rules. The collection is constantly modified such as added, removed or changed. When a rule engine is being used to carry out business rules, there needs a way to identify the suitable business rules. Business rules represented as XML

are ideal for this kind of situation. If business rules are stored as one large XML document, XML-based tools can be used to search for specific rules. If individual rules are stored in separate XML files, then these textual snippets can be stored in a relational database and searched using the own mechanisms of database. In either case, XML reporting tools can be used to display the rules found by a query.

6.2 Semantic Business Rules Transformation

As the advantages of representing business rules in XML format have been recognised, how to correctly and effectively transform business rules becomes a pressing issue. Unfortunately, the characters of natural language lead to difficulties in directly converting raw business rules into standard XML format. Therefore, the best solution should involve a third-party representation as an interpreter tool to help the transformation. According to this understanding, the proposed approach in this chapter divides the whole business rules transformation process into two phases as shown in 0.

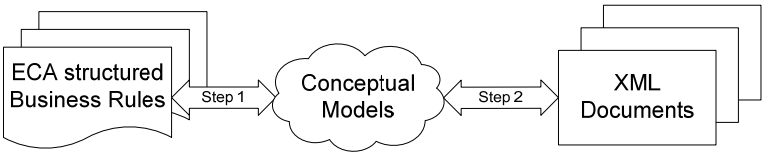

Figure 6-1. Business Rules Converting Mechanism

Step 1 Translate the ECA structured business rules into conceptual models that offer comprehensive and understandable artifacts.

Step 2 Analyse elements in the conceptual models and map them to the target XML schemas.

The reason for utilising conceptual models as intermediate design documents instead of logical models is for database neutral. Indeed, business rules work as metadata to describe application data in information systems. It usually resides in many places such as a document, a database management system (DBMS) or a data repository. Choosing which database depends on the implementing environment of business rules. Utilising conceptual model can build a support useful to database designers working

with any types of databases, as well as to designers that need to integrate business rules in frameworks that are not based on DBMS. Furthermore, representing business rules at a conceptual level also guarantees the semantic of business rules unchanged during the transformation.

To this aim, Entity Relationship Diagram (ERD) provides the best conceptual-level solution because of the following reasons:

Reason 1: ERD has an intuitive and widely known graphical representation. It offers a very good high level documentation on the data structures that cannot be expressed using logical models.

Reason 2: ERD is easy to understand not only for software engineers but also for business professionals. Therefore, ERD can serve as a good communication tool between technical and non-technical people during the process of business rules transformation.

Reason 3: The generated diagram can be combined with other ERDs to integrate the new structures with preexisting data. This integration is much more easy to study at the entity relationship level using the associated graphical view than, e.g., on the flat relational model.

Reason 4: The choice of the ERDs as the intermediate conceptual models allow mapping business rules information to XML Schemas in a natural way, i.e., obtaining an ERD that is XML compatible and that preserves all the information originally stored in the raw business rules. This leads to a mapping process that can be documented and certified soundly and completely.

6.2.1 Modeling Business Rules

There is a critical need for devising rules or guidelines for converting natural language business rules into ERDs. The process of representing business rules in ERDs is called business rules modeling, which is part science and part art. It is part science because there are regulations that show how ERDs should be built, and it is part art

because it requires extracting complete and correct information from business rules written in natural language. The purpose of modeling business rules is twofold:

a) To cast the business rule elements and their relationships into terms with a rationalised version of ERD notations that are transferable to XML documents.

b) To show how business rule elements are related and capture their relationships targeted for a business rule system. Thus, software engineers can obtain a set of semantics-preserving operators for manipulating ERD structures

The business rules modeling will result in the creation of a logical ERD and the creation of all the data elements into data repository. In addition foreign key links, triggering operations and other stored procedures would automatically populate the appropriate repository areas.These purposes suggest a guideline for modeling business rules, that is first identifies the semantics of rules execution and then presents the rule objects in ERDs.

6.2.2 Business Rules Metadata

Modeling business rules in ERDs uses a different set of notations than the ones used in business rules templates. Rule template elements have to make reference to ERD notions, principally rule entities and their attributes. Therefore, how to accurately represent business rules in ERDs is then becomes a very important issue. The key lies on looking at business rules from metadata perspective.

The classical definition of metadata is “data about data”. In more details, metadata is about the correct interpretation of data and how this affects the business as a whole. In business rules circumstance, metadata provides the context and meaning of the data represented by business rules. By capturing business rules as metadata, ERDs can consistently deliver effective strategic information

In some simple business rules, metadata is fairly straightforward and simple. However, as business rules became more complex and varied, there is no clear distinction between data and metadata. Roughly speaking, information that a business

user can manipulate is data, all the other information that a system stores and manipulates is metadata. Data is stored in information system database. Metadata is stored in a special database that is by convention called a repository.

Some simple examples of metadata are:

- Business term or name
- Business definition
- Business valid values or domain
- Data length, precision and type
- Whether the entity is a supertype or subtype
- Estimated number of occurrences (minimum, maximum, average)
- Business steward (role that can change the metadata)
- Primary key
- Alternate keys
- Foreign keys

Since the candidate business rules are represented in natural language, many types of metadata which embedded in such business rules are function as a subset of natural language. In order to making meaning explicit and providing definitions to business terms, data elements, acronyms, and abbreviations. A dictionary or glossary is part of business metadata, Business rules events and actions are treated as fundamental semantic objects, called entities. To achieve this target, a critical success factor is understand every part of business rules model from a metadata perspective.

Each business rule has its basic metadata stored in it. Each business rule must be identified with a rule type. Business rules may be any of the following types:

Type 1. Definitions of business terms

Type 2. Relationships among facts about the business

Type 3. Processing sequences

Type 4. Logical inferences

Type 5. Mathematical and functional derivations

Type 6. Data integrity constraints

Although business rules take different types, there are few notations available for representing business rules in all their subtlety. These notations include:

- Entities
- Attributes
- Relationships
- Connectivities
- Constraints

Since each type of business rules have their specific specifications, a same notation used in different type of business rules may represent different meaning. For ease of identifying counterpart notations in ERDs, software engineers need to classify business rules according to the different patterns they are used.

Based on the understanding of the business rules model, the captured business rules contains a set of logical model elements and are implemented as database tables. Whatever tools are used, the final aims are similar to allow business rules to be defined and implemented as metadata so that information systems can easily store them and use them flexibly.

As a result, a set of metalinguistic features are designed to provide a formal, expressive, and easy-to-use tool for expressing transformational strategies as meta procedures that can be used repetitively on similar occasions.

6.2.3 Classifying Business Rules

Business rules are metadata that used to describe application data. Theoretically speaking, business rules apply to data structure. The widely used object role modeling can then express a few kinds of business rules. But even it does not go all the way when it comes to describing business rules. A solution for this problem is classifying all business rules according to their specification and then applying different modeling strategies to each type of business rules.

6.2.3.1 Term Model

A term is the application of a single definition to a word or phrase. The kinds of terms that usually found in business rules include:

- Entity --- a thing of significance to business, about which information is to be held.
- Attribute --- a descriptor of an entity class.

An entity is an atomic concept or object in the considered domain constructed by a unit of essential information, such as entity name and attributes.

For example, in a customer relationship system, customer is the basic necessary element.a specific object of interest to the business area. Companies might say that in an accounting system, two things are basically necessary: Accounts and transactions. In a customer support system, people would need to have some information about customers, for a start. Each such. The unit of essential information of customer is an entity class. Each entity class has a name and normally one or more attributes. Attributes may be thought of as smaller pieces of information within an entity. Together they describe an entity to the degree that is found necessary. Figure 6-2 are examples of both entity class and attribute names.

In the figure an entity called “Entity Class A” is graphically depicted through a rectangle ERD that contain the entity name. Entity Class A has four attributes that

people have found to be important to know about the customer: customer ID, type, value and customer name. Most likely, the customer will have several other attributes as well, but that will show up during the analysis phase. The attributes in ERD can have symbol # and * in front of them. Symbol # means that the attribute is (part of) the primary key for that entity. The primary key is a unique value for every new customer the company gets so that company can uniquely identify him from all the other customers. The attribute ID has a * in front: This indicates that it is a mandatory attribute; for every new customer, ID must be filled in. The opposite is also possible; if an attribute has an "o" in front of it, it is an optional attribute, and a value is not required.

6.2.3.2 Fact Model

A fact is the attribution of something to describe a thing: a role it plays or some other descriptor. There are three kinds of facts can be shown on a data model:

- Relationships (roles) -- A has a business relationship with (plays a role with) B.
- Attributes -- A is an attribute of B.
- Super-types/Sub-types -- An occurrence of A is also an occurrence of B.

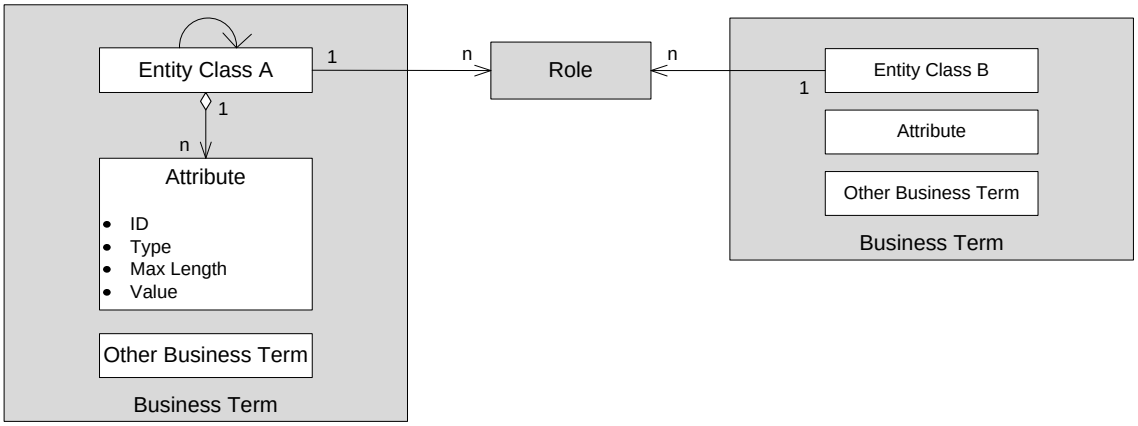

Figure 6-2. Meta-Model of "Fact"

Meta-model in Figure 6-2 describes the nature of "Fact". A relationship between entity classes A and B consists of two roles --- one going each direction. The model

shows that each entity class may be connected via one or more roles. Each of these, in turn, must be connected to exactly one other role. This second role is then connected to another entity class. The fact that each attribute must be about a single entity class is shown by the relationship between attribute and entity class A. In the same way, the fact that one entity class may be a sub-type of another entity is shown at the top of entity class A. Such facts can also be shown on a data model, as evidenced by the fact that attribute and entity class are themselves sub-types of business term in this drawing.

6.2.3.3 Derivation Model

A derivation is an attribute that is derived from other attributes or system variables. Figure 6-3 shows the meta-model of the derivation logic. Note that the derived attributes can be shown on the data model, but the derivation formula cannot. In UML, it is possible to add a note to the diagram to include the derivation formula.

In the model, each attribute may be computed via one or more derivations (formula), each of which must then be composed of one or more derivation elements. Each derivation element, in turn, may be either the use of an attribute, a constant or the use of a system variable. Each derivation element specifies the operator to be used in manipulating the element, and, optionally, a role operator to apply if a role is used.

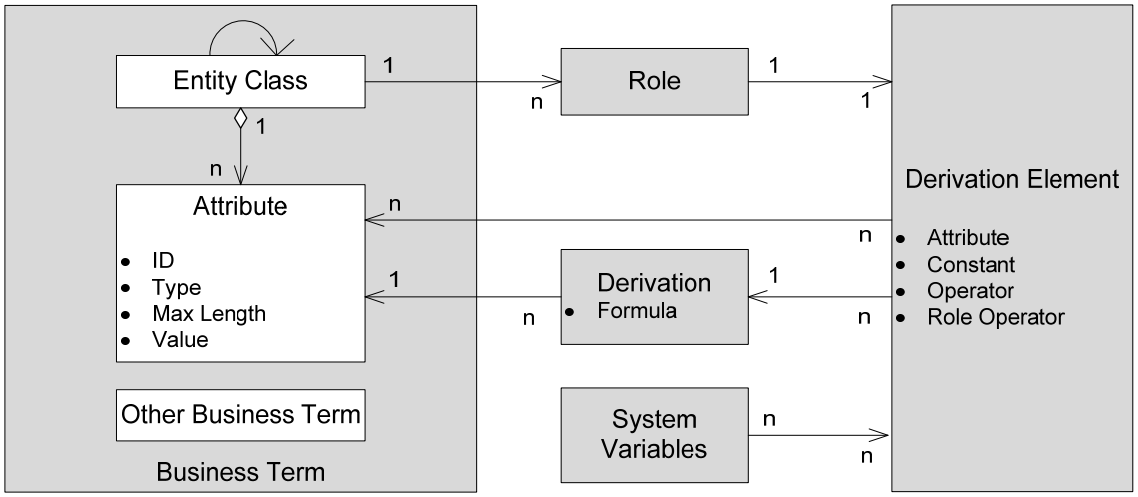

Figure 6-3. Meta-Model of "Derivation"

6.2.3.4 Constraint Model

The business rules model describes the kinds of business rules and takes up those that can be represented in an ERD. This includes terms, facts, and derived attributes. the fourth kind of rule (constraints) and the difficulties in representing these

Although business rules take different types, they all generate based on ECA theory. The basic structure for all business rules should be same as a single, coherent business rules model and every business rule is an instance of that model. In practice this means that each business rule belongs in only one place in a business rules repository; similar business rules can be grouped together; and these groupings may be related by sharing common pieces of business rules. These objects are discovered, recorded and structured into a design by a process called business rules modeling.

Business rules modeling can constrain how a business rules repository should be built This model is typically built using business rules design practices based on the concepts of ECA theory. Business rules model provides a visual representation that enables software engineers to design a blueprint for business rules repository which is used as a component in BRM subsystem. Modelling business rules can also let software engineers see what a business rules repository looks like before it is physically implemented. In particular, the structure of business rules must be understood by business rules engines that use them.

Thus, business rules model can be regarded as an example of metadata that are used to describe business objects and facilitate the integration of business rules. In general, business rules model can be defined as metadata, modeled as metadata and implemented as metadata.

Business rules put a whole new spin on the process of understanding what is going on in a business. Modeling the metadata of business rules allows us to see just what the nature of that spin is.

6.2.4 Converting ERDs into XML

As business rules been modeled, there need a business rules language to specify the

syntax and semantics of business rules so that business users can take part in the business rules development process. The requirements of that desired business rules language include two aspects:

1. Based on an independent format. Business rules engines are the main consumers of business rules. Currently, there exist many commercial off-the-shelf rules engines in the market. Most of them use business rules in proprietary format and no mapping systems exist for any pairs of them. Since information systems often integrate rules engines from different vendors to gain more competitive, when they change the used or add new rules engines, they have to spend much energy on rewriting business rules. In order to let various business rules engines share the same rules source, the desired business rules language should independent to any of the rules engines.

2. Support web service technique. The target information system is built on SOA. Business rules language response for communicate and exchange business information. It is necessary for the defined language to be implemented as web service so that they can be correctly interpreted by service suppliers (rules engines).

The idea behind these requirements is to find out a unified business rules language which can be easily implemented in SOA. XML is such a language that can accomplish this task. Just as SOA is a universal software architecture for building information system, XML can be seen as a universal data-structuring architecture for the system. To use business rules in information systems, software engineers will want to embed them in a service-oriented application; likewise, software engineers will want to represent the rules themselves in XML.

In order to represent business rules in XML format, software engineers have to develop XML DTDs to describe the new XML type, DTDs are defined in a kind of pidgin XML --- a language that is similar to XML. `ELEMENT` lines describe individual XML elements and their contents. `ATTLIST` lines list the XML attributes that an element is allowed to have. A list of rule elements is called a `rulebase`; a `rule` element has a `name` attribute and optionally a `priority` (salience) attribute. The content of a `rule`

consists of an `lhs` element and an `rhs` element:

```
<!ELEMENT rulebase (rule)*>

<!ELEMENT rule (lhs,rhs)>

<!ATTLIST rule name CDATA #REQUIRED priority CDATA "">
```

There are two broadly different strategies for representing rules in XML. The most general strategy is the one taken by the RuleML projects. In this strategy, an XML element represents the concept of a pattern and the contents of the pattern are data inside the element. For example, in the business rule

"`Discount will be given to the customers whose purchase is over 1000.`"

The fact pattern in this rule

```
if purchase 1000
```

is represented in RuleML as

```
<fact>
    <atom>
       <_opr>
           <rel>if</ref>
       </_opr>
       <var>purchase</var>
       <ind>1000</ind>
    </atom>
</fact>
```

The interesting thing about this representation is that the data could change and the elements themselves could stay the same. For example, the pattern

```
if purchase 2000
```

would look like this in RuleML:

```
<fact>
    <atom>
       <_opr>
           <rel>if</ref>
       </_opr>
       <var>purchase</var>
       <ind>2000</ind>
    </atom>
</fact>
```

This means that rule editors and all rule-processing software can work with this kind of pattern without knowing anything about the meaning of the data. The metadata about the pattern is explicit; indeed, it is the important part. This approach is powerful in its generality. The downside to this kind of rule language, however, is that it is verbose, and that some of the software needed to work with it is fairly complex.

These requirements lead to a standard language for business rules representation. If all applications used standard business rules to capture business rules, it would be much easier to switch application vendors and migrate the rules to the new application. Standardise business rules language can reduce the cost and investment risk for consumers because consumers are not tied to a single producer. Standard business rules would let all the rule-enabled applications share a common rule repository so that all rules can be viewed and modified in one place. The desired format should also allow rules to be modeled fully, including specific values for metadata. These values should then migrate automatically to any database design generated from the models and, ultimately, be automatically inserted into the database tables when they are created or altered from the design.

The approach taken to transform business rules into XML format relies heavily on utilising metadata that exists for business rules. Such metadata contains business-relevant information and represents much of the raw material that a business rules repository must utilise.

6.2.5 Expose the Ruleset as a Web Service

Rules engines such as JRules provide tools to generate wrapper Web services or Session Beans to wrap a newly developed ruleset. Web services developers will be instrumental in creating a wrapper to expose ruleset as a Web service.

XML is a key standard for integrate rules, EJBs, BPEL process flows, and Web services. BPEL uses XML natively to access data while Web services use it to serialise data (and will use it unmodified in Doc/Literal invocations). XML can be used directly in rules. By marshalling, XML can be transformed directly into a JAXB object tree. Rules can be executed with native Java Objects.

Web services should import XML Schema in their respective WSDL definitions. Generated DTO objects from XML Schema (e.g. JAXB) can also help ensure that data is translated smoothly without conversion errors.

Eligibility Web service provides the translation from XML to JAXB and then invokes the Eligibility Rules Delegate Session Bean. To hide the complexity of invoking JRules custom libraries, developer would create a session façade. This approach makes the implementation rule engine "agnostic;" the system could be easily ported to a new rule engine provider. Eligibility Delegate Session Bean makes an RMI call to Eligibility façade Session Bean. This session bean invokes the Eligibility ruleset in the RuleApp package using ruleSession.executeRules("RuleApp/eligibility").

The following List 6-1 is an example of business rule from business rule repository

```
  <?xml version="1.0" encoding="utf-8" ?>
- <Rules desc="Demonstation of Rules Processing Engine"
author="Jeff Bramlett">
- <Rule id="SetupRules">
- <Action type="run" actionId="SetupValues">
        <Match />
        <Assign />
 </Action>
</Rule>
- <Rule id="SetupValues">
- <Actions>
- <Action type="makeVar" actionId="Qty">
  <Match />
  <Assign>0</Assign>
     </Action>
- <Action type="makeVar" actionId="Weight">
        <Match />
        <Assign>100</Assign>
       </Action>
- <Action type="makeVar" actionId="Price">
     <Match />
        <Assign>100</Assign>
     </Action>
</Rule>
- <Rule id="CalculateInvoice">
- <Rule id="CalculateWeights">
- <Rule id="SetDiscounts">
- <Action type="var" actionId="StarQtyDiscount">
- <Match>
- <![CDATA[ StarQty > 50 ]]>
```

```
 </Match>
  <Assign>.3</Assign>
</Action>
</Rule>
- <Rule id="CalculateTotalPricing">
- <Actions>
- <Action type="var" actionId="TotalPrice">
        <Match />
        <Assign>(StarQty * StarPrice) - ((StarQty * StarPrice)
* StarQtyDiscount)
</Assign>
      </Action>
- <Action type="var" actionId="ShippingTotal">
        <Match />
        <Assign>ShippingRate * WeightTotal</Assign>
      </Action>
- <Action type="var" actionId="InvoiceTotal">
<Match />
<Assign>(PriceSubTotal - CustomerTypeDiscountAmount)+
SalesTaxTotal + ShippingTotal</Assign>
      </Action>
    </Actions>
- <Triggers>
    <Trigger id="TriggerOne" varID="InvoiceTotal"
ruleID="TriggerTest" />
    </Triggers>
</Rules>
```

List 6-1. An Example of Business Rule in XML

6.3 Summary

This chapter expresses the transformation strategy from ECA structured business rules to XML formatted representation. The obtained raw business rules can greatly help people to understand the business knowledge embedded in legacy systems. Therefore, a business rules representing language is needed to specify the syntax and semantics of business rules for software applications.

Business rule modeling provides a visual representation that enables software engineers to see what a XML-based business rule looks like before it is physically transformed.

This chapter provides a methodology of translating the conceptual schema of a

relational database into XML schetna through ER model. Physical data are then translated from relational table to XML document. The semantics of the raw business rules, captured in ER diagram, are mapped to XML schema using stepwise procedures. The physical data are then mapped to XML document under the definitions of the XML schema.

Chapter 7
Service-Oriented Migration

Objectives

- BRM subsystem components analysis.
- To show how to migrate legacy systems into service-oriented platform.
- To show how to combine different modelling techniques for workflow integration.

Bussiness Rule Management (BRM) subsystem is the practical art of operating information systems based on business rules approach. Adopting BRM subsystem in information system project can decrease development costs and dramatically shorten development and maintenance cycles. In this chapter, a novel BRM subsystem will be introduced. The key advanced feature of this proposed BRM subsystem is integrating web service technique with business rules approach. All components of service-oriented BRM subsystem are implemented as service suppliers or consumers. In this way, the service-oriented BRM subsystem can easily to deploy and integrate new business requirements.

7.1 Business Rules Lifecycle

All of the information system activities are defined and constrained by business rules. BRM subsystem empowers business users with tools and skills to help them manage their own rules. When looking at business rules, it is important to address their entire lifecycle, because that will be helpful in understanding the architecture of BRM

subsystem and its rule manipulating mechanism.

Business rules lifecycle is an essential technique need to incorporate in the BRM subsystem development. Rules will be conceived, built, operated, and eventually, stored into repository. This flow mechanism enables system designers to specify a particular order to carry out the knowledge modules or tasks. Referring to the business rule lifecycle, BRM subsystem can be developed to deliver complete rule management.

Business rules lifecycle spans all important phases from acquisition through deployment to termination. It composes of five basic stages: rule definition, rule storage, rule deployment and configuration, rule enforcement and rule validation. The business rule life cycle is illustrated in Figure 7-1.

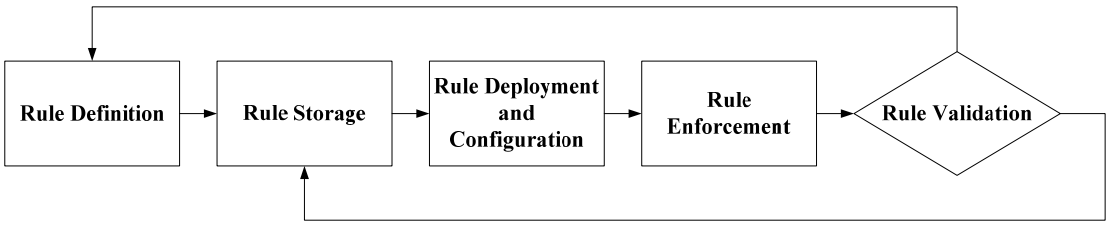

Figure 7-1. Business Rules Lifecycle

Rule Definition is the phase that a rule is created, browsed and validated. During the definition phase, enterprise objectives and constraints are discussed, raising issues that need to be solved in order to generate operational business rules. This phase includes not only the acquisition of business rules, but also captures the deliberation process that lead to those rules, starting with high level enterprise goals. Corresponding definitional tools such as editor, browsers and verifiers can be used by business analysts to input the different rules that are to be active in the BRM subsystem.

Rule Storage is the phase when a business rule is put into persistent storage. All of the rules include inactivated rules and the rules which are ceased to exist in the system are stored in this stage. For BRM subsystem, a rule repository is usually required as the placeholder for storing business rules.

Rule Deployment and C*onfiguration* is the process that rule repository response for

the rule service enquires by deploying the specific rules into the target applications and configures the rule parameters correspondingly and waiting for further execution. During the deployment phase, the methodology distinguishes between the deterministic and nondeterministic business rules. The deterministic rules uniquely characterise the situations where they can be applied without further decision making. The decision is made at analysis time. Therefore, when the rules are encoded in enterprise information system, they will automatically be applied by the system without human interaction; or they will strictly be applied by humans. A set of automated configuration utilities will usually simplify the tasks performed in this phase.

Rule Enforcement is the stage when a rule is being enforced to govern and constrain the behavior of target systems. This process may also affect related decisions. The information resulting from the analysis of monitoring data will illustrate whether the chosen rules achieved the supposed goals or not. Monitoring and reporting tools enable rule makers to understand how the status of business rules enforcement and whether the policy has been defined reasonably.

Rule Validation occurs throughout the period of rule enforcement and last until analyser draws a final conclusion. To guarantee the deployment quality of business rules, analyser is used to monitor the consequences of the decision by identifying observable information which can be captured during operation. This information will measure the effects of applying a business rule and will validate or deny the decision of choosing that particular business rule after reflecting on the monitoring data.

These stages of rule flow compose the whole lifecycle of business rules. The proposed BRM subsystem is built based on this rule flow mechanism; each stage corresponds to one component in the subsystem. This modularisation character will help BRM subsystem cope with rapid changing requirements.

7.2 Features and Architectures of BRM Subsystems

BRM subsystem, as a whole, can be viewed as an ecosystem composing of various components. Its framework is based on the mechanism of business rules lifecycle.

7.2.1 Features

Different from the component-based development which scales up the object-oriented philosophy to the component level, the development of service-oriented BRM subsystem makes a further extension by separating coherent business objects into a set of components, each of which supplies a part of rule services. The service-oriented BRM subsystem follows web service principles to make application resources more flexibly available throughout information system. This leads to the system being built from a set of loosely connected components. Applications therefore can deploy themselves as services, many of which are ripe for reuse, or even better–sharing.

In order to achieve these advantages, the desired BRM subsystem should have the following features:

- Allow business analysis to create and modify the rules;
- Use a fully-featured repository;
- Support backward chaining;
- Allow the rule engine to be a component or service within subsystem;
- Allow application to be deployed in a service oriented architecture;
- Focus on business rules management problem;
- Provide good report generation facilities;
- Be compatible with a component-based or service-oriented architecture;

7.2.2 Architecture

BRM subsystem is not necessarily a monolithic application. Based on the principle of SOA, BRM subsystem is more of an approach. It is a set of services that can be implemented to a greater or lesser degree in any number of situations to meet a precise set of requirements. The proposed BRM subsystem strongly advocates taking advantage

of using whatever components of rules engine design apply to specific circumstances.

Figure 7-2 gives an overview of the components in the proposed BRM subsystem architecture. It is mainly composed of four independent components (grey rectangles): business rules editor, business rules repository, business rules engine and business rules verifier. These components are made available as hosted rule services respectively within BRM subsystem and co-operate using standards web service techniques.

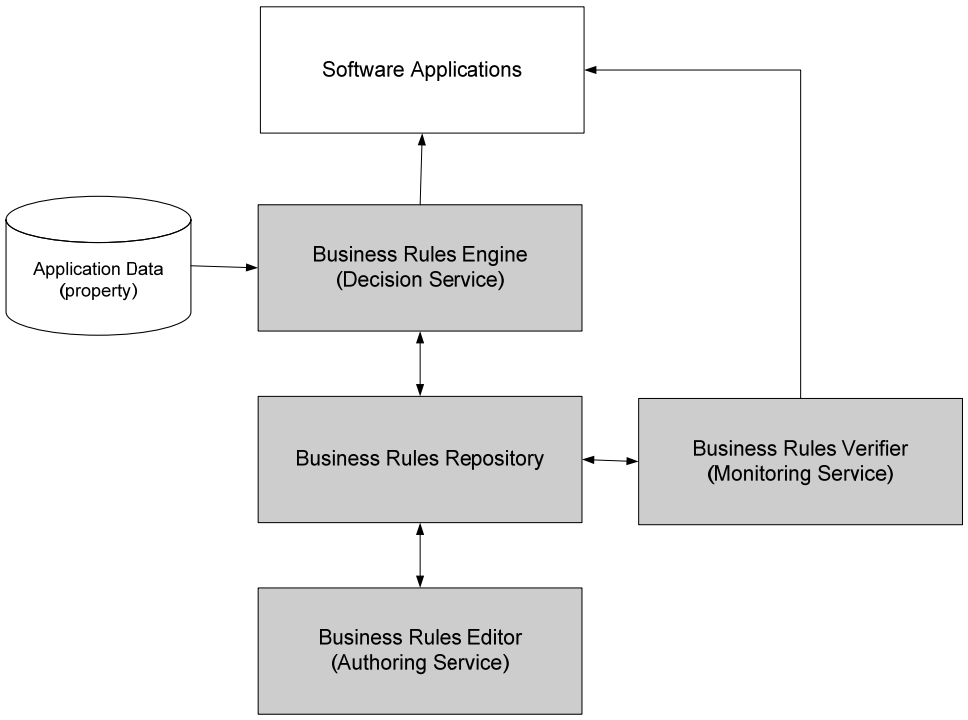

Figure 7-2. BRM Subsystem Architecture

The architecture illustrates the process of modular units cooperate to provide inference services for rule service customers. BRM subsystem also uses a separate rule authoring package for rule definition and editing.

The BRM subsystem operation phase begins with a Business Rule Editing Component which creates the initial rules in an integrated development environment. In service-oriented environment, rules editor is made available as a hosted web service through implementing XML standards and DOM functions. The combination of these two techniques gives prominence to non-technical business users; helps them quickly master the rule editing method. This novel rules editor represents an unprecedented

level of flexibility in business rule authoring metaphors and customisable user interfaces for developers.

Before implementing business rules, a Business Rules Repository is needed to be set up. Business rules repository is a common location for storing rules and rule metadata. It is based on standards XML format and provides a foundation for metadata exchange with other repositories. The rules repository is designed to be customisable and extensible for each unique requirement from organisation. Moreover, the central located character of repository makes rules can be managed and deployed in applications throughout the enterprise. The architecture emphasises the separate character of the repository (from which rules may be imported into rule engine) and the probable presence of a database or application data that must be coupled with business rules in software applications.

The next part is named Business Rules Engine. The business rules engine is the main parts responsible for providing decision making services. It utilise mechanisms to apply the knowledge in a rational manner to the problems; chains the rules together to reach valid conclusions. In the BRM subsystem, the business rules engine component is operated as an independent program. This is because in the real business environment, it is very difficult to write a single list of instructions covering every situation that might possibly arise; even worse when considering that circumstances interact to constantly change the correct outcomes of each decision. BRM subsystem makes use of the abundant supply of existing commercial rule engines to cope with various situations in a real business environment. As a result, an external rule engine aided BRM subsystem can easily and efficiently adapt with any complex systems by simply changing the integrated commercial business rule engines.

Business Rules Verifier monitors the execution of business rules It watch over the critical services of the business rules engine, provide reports to help business professionals understand how the status of rule enforcement and whether the rules has been defined reasonably.

7.3 Components Description

The proposed BRM subsystem allows various roles within an organisation to author, save, deploy and interact with business rules in a centralised manner that allows disparate applications to consume the rules in a managed, service-oriented manner. As such, BRM subsystem provides a rule editor which allows users to author and save business rules; a shared rules repository for persisting and looking up rules; a high-performance rules engine for executing rules in the repository; and administrative tools for deploying, retiring and migrating rules.

7.3.1 Business Rules Repository

The design scope of rule repository covers the definition of tables and the relationships between them. Business rule repository adopts property driven approach to specify business rule entity and reveal the extended rule entity relationship. Figure 7-3 illustrates ER diagram of BRM repository.

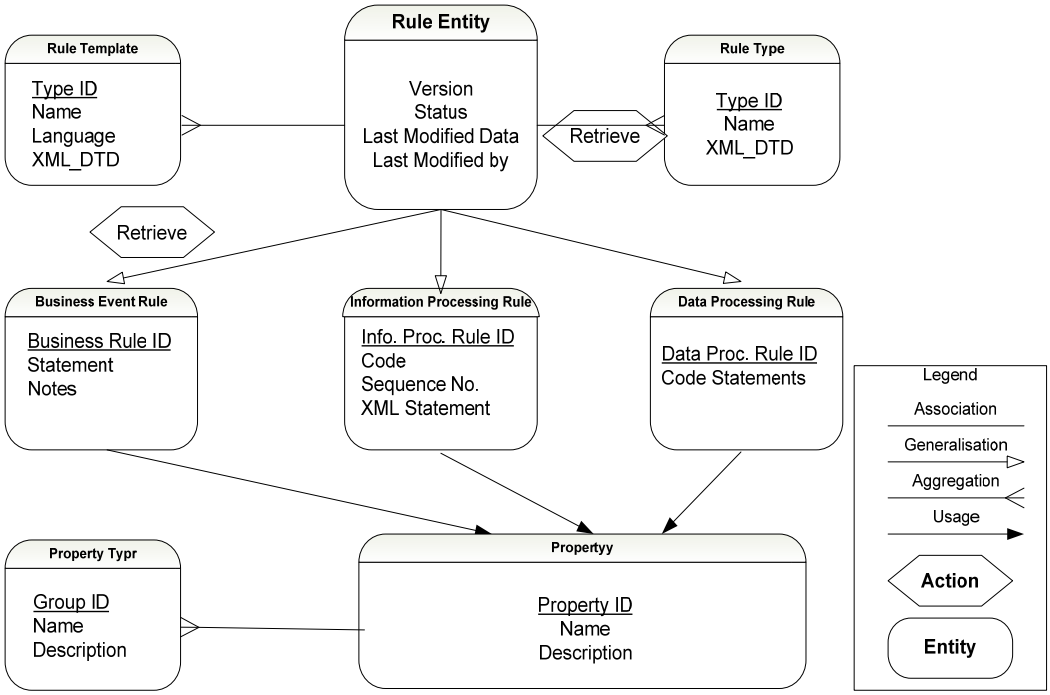

Figure 7-3. E-R Diagram of RBM Repository

Business rules repository is a database that stores business rules metadata. The major use of BRM subsystem in information system is to manage business rules, rather than to perform tasks that do not deal with stored information. Thus, business rules repository occupies a central role in BRM subsystem design. In particular, the structure of business rules repository must be understood by the business rules engines that use them. Tree structure repository is more suitable. The design of tree structure allows an organisation to have a central parent repository with all common data elements of the enterprise. Child repositories inherit the global definitions. Locally defined data elements will become only global if they are needed by at least two entities in the organisation. Another use of the tree structure repository is work-inprocess. Specifically this means that new projects may create potentially new data elements to be added to the global data repository. This allows organisation to properly control any new entries into the global data repository until the project is completed or the data element candidate is approved by the appropriate people in the organisation.

The major task of BRM subsystem is to mange data, rather than to perform tasks that do not deal with stored information. Thus, business rules repository occupies a central role in BRM subsystem. In particular, the structure of application databases must be understood by the rules engines that use them.

Business Rules Repository defines all of the necessary attributes for a rule that will allow it to be effectively introduced, processed and retired with minimal intervention. Since BRM subsystems are using an XML-enabled repository, users of business rules can query the system using a language like XQL to find out what business rules are applicable in a specific situation.

Business rule repository is used to store business rules generated by rule definition component and support business rule maintenance and management. The *rule decision point* is responsible for interpreting the rules stored in the repository and communicating them to the rule enforcement point.

From the SOA angle, a rule repository can be regarded as a rule definition services. This character has two benefits as below:

- Supporting developers to work in a coordinated manner as teams and leverage work by sharing and reusing rules, rule sets, rule flows and object models.
- Querying and reporting capability for impact analysis, traceability and business rule reused by SOA-based applications.

7.3.2 Business Rules Editor

BRM subsystem must be flexible enough to accommodate rapidly changing business requirements. Often, this requirement is met by allowing business users to write and modify business rules. Business rules editor is used to displays business rules and allows rules to be written and modified by nontechnical users.

Currently, there already have many rule engines that provide integrated rule editor or a toolkit for developing rule editors. However, these editors are only designed for developing rules in the native language of the rule engine, not in vendor-independent XML format. As the aims of the proposed BRM system is operating in an independent manner and keeping interoperability, a requirement is aroused to develop a customised rule editor specifically for standard XML based business rules.

The rule editor that used in the specific BRM system is a tool for creating and modifying XML files. Figure 7-4 illustrates the structure of XML rule editor:

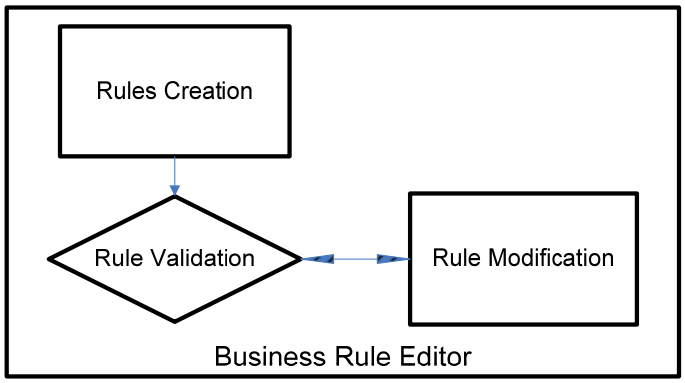

Figure 7-4. The Structure of XML Rule Editor

In fact, the business rule language can be customised to support the specific application. The tool or tools should allow rules to be modeled fully, including specific values for meta-data. These values should then migrate automatically to any database design generated from the models and, ultimately, be automatically inserted into the database tables when they are created or altered from the design.

This means that a business rule can be documented once in a logical model and still be part of multiple system designs and implemented databases. This one-logical, many-physical representation of business rules as meta-data also allows a significant change in software component design. Components can be designed to access database tables for rules and do not have to include complex decision tables and rule-based processing logic.

Keeping the housekeeping information up to date can become a bit of a chore. If it is not an attractive job to do, the chances are it won't get done well, and some parts may not get done at all. The editing feature of Service Oriented BRM model will cover this shortcoming and enable business users to manage business rules separately.

7.3.3 Business Rules Engine

There are a number of popular open source business rules engines like Jess and Drools. They could be combined with a separate repository. Business rules engine is software application that utilises business rules to make inferences. Users may want to know something, and the rules engine can generate an answer to their question based on rules definitions that it possesses and data that it can access. Alternatively, a user may simply wish to know what can be inferred from a given set of data based on the rules contained in the engine without necessarily posing a specific question to the engine. Rules engines oriented to inference in this way are related to applications that are called expert systems or artificial intelligence.

There are literally hundreds of business rules engines that claim to embody business rules. But if filter the list according to those that have a moderate amount of features and traction, either commercially or in the open source community, there get the

following alternatives:

- BizTalk
- Blaze Advisor
- Jena
- Jess
- JRules
- OpenRules
- PegaRules
- RulesPower
- Drools

Quite a few rule engines are available that are suitable for use in enterprise applications; Each rule engine has its own strengths and weaknesses, In this use Jess and Drools as two example rule engines for the prototype implementation.

The following example is by Said Tabet, who has written an XSLT style sheet that converts a subset of RuleML into Jess rules (List 7-1):

```
(defrule AnimalsRule1
   (declare (salience 10))
   (has ?x hair)
   =>
   (assert (isa ?x mammal)))
```

List 7-1. Jess Rules

Here is the same rule in RuleML (List 7-2):

```
<imp label="AnimalsRule1" priority="10">
   <_head>
      <conclusions>
         <assert>
```

```
         <fact>
            <atom>
               <_opr>
                  <rel>isa</rel>
               </_opr>
               <var>x</var>
               <ind>mammal</ind>
            </atom>
         </fact>
         </assert>
      </conclusions>
   </_head>
   <_body>
      <and>
         <fact>
            <atom>
               <_opr>
                  <rel>has</rel>
               </_opr>
               <var>x</var>
               <ind>hair</ind>
            </atom>
         </fact>
      </and>
   </_body>
</imp>
```

List 7-2. RuleML

It is obvious that the RuleML version is much longer. This is typical of most uses of XML; the reason is simply that the XML contains more information. The individual tags explicitly state the purpose of each component of the rule, whereas in the Jess format, that purpose is implicit in the syntax.

In order to locate the appropriate business rules approach within information system, Service-oriented BRM subsystem provides a way of managing the decision logic centrally and independently. The key is the use of a repository to store the rules and a rule engine to apply them correctly in the context of a particular application or service. Figure 7-5 shows the configuration of this structure.

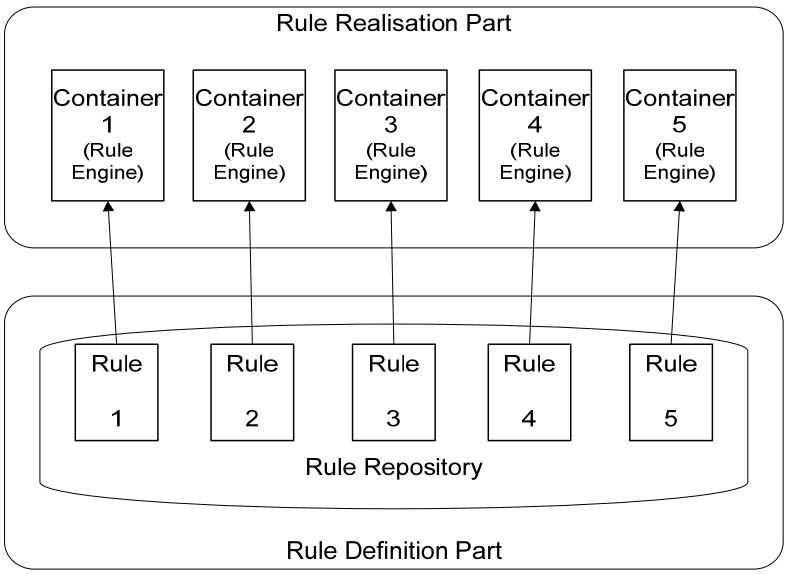

Figure 7-5. Structure of Business Rules Engine

This architecture designed to adopt various existing rule engines for improving its flexibility in different environment. Each of these rule engines works as an independent supplier to provide rule realisation services. Instead of communicating directly with the repository, a rule enforcement component uses an intermediary known as the *rule realisation point*.

The responsibilities of it can be generalised into three aspects: Reactive Management, Deliberate Management and Reflective Management.

In fact, most of the existing rule engines have combined more or less rule definition function within rule realisation part. However, no matter how wonderful the rule engine is, it is not realistic to put absolutely everything in it. Moreover, most rule engines ignore the existence of any other kind of business logic. If a required logic outside the rules engine is deemed not to exist, user will be forced into a tortuous realisation of a rule within inadequate rules engine, because they are not be able to have a rational discussion about whether a rule should be implemented inside the rules engine or outside.

The proposed Service-Oriented BRM system take advantage of the combination of rule definition and realisation in rule engines, treat them as a kind of container that

implements rules in a particular way. In this circumstance, business rules can be bound to BRM systems at two points of its lifecycle:

a) *Rule Deployment & Configuration.* This type of binding is called early rules binding between rules and BRM system since it is realised at the preparation time. In this phase, rules are active in the rule definition part of the rule engine but have not been committed to any business activities yet. The rule definition part is act as a cache in the whole system to help compensate for differences in the deployment rate of business rules.

b) *Policy Enforcement.* This type of binding is, on the other hand, called late binding between rules and BRM system since this binding is realised at the run time when business rules are being executed.

Since rule engine working alone is not sufficient, there needs a rule repository to cover all the angles described above. Rule Repository stores the business rules in a layer separate from rule engines. In a real organisation, an independent and centralised rule repository in the information systems will offer obvious maintenance advantages and makes rules more readily reusable.

One important use for rule engines in an enterprise environment is to implement *business rules*. In the service oriented architecture environment, the enforced business rules are regarded as rule services that support decision making processes. A set of automated configuration utilities will usually simplify the tasks performed in this phase.

In SOA-based applications, the BRM system management operation is performed through the Business Process Management (BPM) system. The Business Rule Engine in BRM system is considered as loosely coupled rule service which provides an interface to execute rules and methods defined in business rule repository. When an event controlled by a rule occurs, BPM system invokes the associated rule service to make application specific decisions. Such a rule service usually receives clearly specified input/output objects and a reference to the business rule sets to be examined during service execution. According to the requirements of the rule services, a rule engine will be selected. The selected engine executes a business rule from the rule repository and

passes it the input/output objects. Then are generated, rule engine will sent the execution results as parameters to the enquiring application through rule service. The Figure 7-6 represents the proposed business rule engine incorporated into a service-oriented architecture

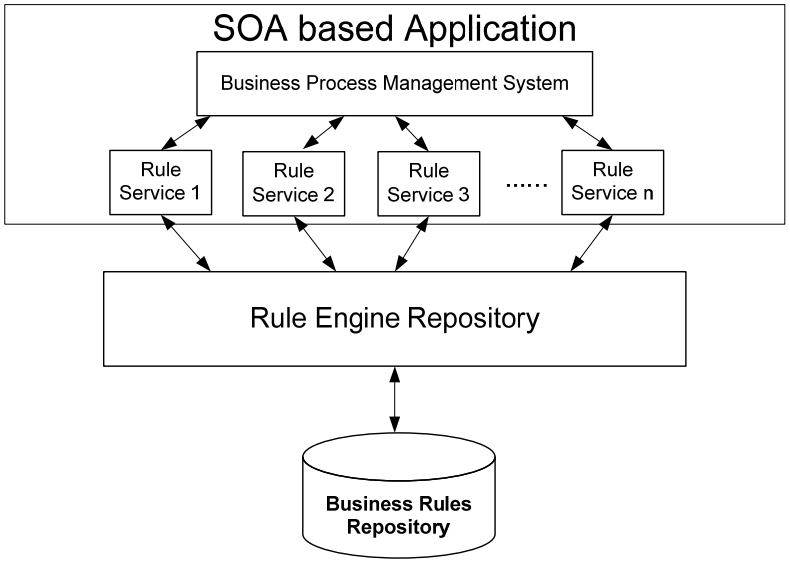

Figure 7-6. SOA-based Business Rules Engine

7.3.4 Validation and Error Handling Criterions

The XML editor will process XML files that are invalid or not well-formed. The business monitor determines all of the information collected from business rules deployment process, every unsuccessful or incomplete business rule will be recoded using their reference number in the rule repository. The monitor will then generate a report which shows the incorrect business rules and pass it to the rule editor. The examine factors considered by editor such as what is missing or whether any changes have occurred in the deliberation information. For example, one could determine that there are some missing criteria, arguments, alternatives or assumptions, or that changes might have occurred in the values of the assumptions underlying a business rule. Also, changes in internal or external sources of an enterprise might induce the necessity of changing some business rules.

7.4 Problem Statements

Current business rule approaches lack a sophisticated way to be integrated within SOA due to a number of current shortcomings.

The first problem is the lack of a remote interface to allow a remote rule invocation from distributed applications. Most rule engines are currently used as a library which is linked to the enterprise applications. The rules are typically invoked via proprietary interfaces provided by the rule engines (e.g., rules for calculating the discount of a customer). The accessibility and reusabilty of these rules is limited to local applications (running on the same machine) although other enterprise applications may need this set of rules for achieving a task.

The second problem is the lack of standardisation of a rule representation language. Each rule engine vendor proposes a custom format for representing their rules, thus a number of different dialects have emerged. This diversity makes it hard to use heterogeneous rule engines in the same enterprise environment or switch to another rule engine. It also hampers using rule engines in distributed environments where other applications and services can use the rule engine as a computational entity.

The third problem concerns the execution of business rules. A remote communication facility among rule engines would allow retrieving data from other rule engines to perform the inference process. Such a process is refered to as *distributed rule execution*. Currently, the lack of such a distributed rule execution can be seen as a consequence of the first two problems.

Pure business rule knowledge approach is not suitable when dealing with inherently object-oriented applications because it lacks the ability to organise the process of system operation. Fortunately, over the last years, different Web Service composition languages have emerged to orchestrate business process, such as the Business Process Execution Language for Web Services (BPEL) and Business Process Management Language (BPML). However, they all fail to apply the objective that integrates the business rules based knowledge into a service oriented system. This section will

introduce an approach to generate meaningful Web Services among legacy systems workflow for accessing and executing business rules. Since the rule engines functionalities are exposed as services, they can be reused across all inter-enterprise application. This making the development of new applications and integrations easier and fulfilling the promise of a service-oriented architecture. In addition, the components built in different programming languages can run on different platforms and communicate with each other across different network systems seamlessly in a heterogeneous service oriented computing environment.

7.5 Exposing Application Logic as a Web Service

To make existing application logic available via a Web service—while achieving loose coupling, one of the most important principles of service-orientation—first will need to decompose the business logic of application into a series of lightweight and independent services to be then utilised within a composite service-oriented solution. It is interesting to note that an SOA solution may be built upon the existing application structures, often without the need to change those structures.

7.5.1 Granularity

When reverse engineers combine/divide legacy functionalities to compose candidates for target system services. One of the most frequent questions that often address during this process is how big a target service should be. The answer is “It depends on the service available and the number of interactions required to accomplish a specific goal”.

An effective target service must be goal oriented and is defined depending on the desired level of service granularity. The concept of granularity refers to the scope of functionality provided by a service. In general, finding the right balance of service granularity is a challenging task that related with the service reusability issue. At this point, reverse engineers can use the knowledge learned from section 0 to construct target services from legacy elements.

7.5.2 Adapters or Mediators

Several approaches using adapters or mediators have been presented to build independent and reusable components by isolating and managing the components' interactions outside the components. There are two types of interaction styles between adapters. The first type requires adapters to communicate through a mediator or a facilitator. The second type requires adapters to communicate directly. One of the disadvantages of having adapters communicate through a mediator or facilitator is decreased implementation complexity at the expense of increased interaction overhead. However, software developers do not have to be concerned with communication issues caused by different data formats, different network types, and different computer architectures. They can focus on the logic within the adapters. Although the direct communication style decreases the communication overhead, it requires software developers to deal with communication issues. In addition, the adapters need to be modified if its components are moved to a new environment in which different platforms or networks are used. From the software engineering point of view, people believe the benefits of using a facilitator or a mediator for component independence outweigh its disadvantages since these types of adapters increase independence.

7.5.3 Web Service Wrapping Model

Wrapping offers an easy and cost effective alternative to replacement. The approach works by hiding the complexity of the system through modern looking interfaces. LIS can be wrapped at the functional level, data level or at the user interface level. While wrapping offers a cost effective alternative, it very much remains a short-term solution for LIS renovation. The approach solves the immediate problem of LIS, but in the long term wrapping means an additional layer of code to manage. Moreover, being completely dependent on technology implies that the layer will continue to grow whenever the current technology is replaced .

7.5.4 Services Mapping

To convert the code into a service based component, an IDL interface is first

created from this reusable code and its related data structures. A service-providing component is then generated using the IDL interface. The remaining code in a program is converted into a service-requesting component in which the common code is replaced with a function call interfacing with a service-providing component. The placement of the functional call should guarantee that the input data is already to be passed to the function and the output data will be timely produced for all subsequently executed statements.

7.6 Workflow Integration

Web service orchestration technologies are emerging as practical means to integrate business partners' applications with higher levels of functionality, negligible performance degradation, reduced development effort, and better quality of service. However, because most compositional languages address distribution concerns at the early stage of the business process definition, Web-based applications become more static and more difficult to develop.

Web services can be seen as basic modular units that can be reused with no knowledge of how their functionality is implemented. Orchestration enables more sophisticated forms of interaction during business collaboration in order to coordinate the activities of the participants. Compositional languages (like BPML and BPEL4WS) are used to describe business processes by providing a set of basic activities and structured hierarchical composition schemas to control the interleaved execution of the basic activities.

7.6.1 BPEL4WS

BPEL (Business Process Execution Language) for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment - even across multiple organisations - using a combination of Web Services. It has built-in support for asynchronous interactions, flow control and compensating business transactions. It integrates with XPath, XSLT and XQuery for XML data manipulation

BPEL4WS defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web Service interfaces, and the structure of the relationship at the interface level is encapsulated in what people call a *partner link*. The BPEL4WS process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. BPEL4WS also introduces systematic mechanisms for dealing with business exceptions and processing faults. Finally, BPEL4WS introduces a mechanism to define how individual or composite activities within a process are to be compensated in cases where exceptions occur or a partner requests reversal.

7.6.2 The Integration Architecture

The proposed approach separates business logic into rules, BPEL, and Web services:

- Eligibility BPEL Process explicitly defines the steps that must be executed based on the received Eligibility Results data. Based on the Eligibility Results, Eligibility BPEL Process will invoke different branchs.
- Eligibility Web services execute medium-grained tasks for the BPEL Process Manager, which encapsulates the mechanics of how to send correspondence and queue tasks. Web services operate on summary data from the common data model such as database keys (such as an Account OID) that can be used to extract detailed data to perform the required task.
- Eligibility rules do not modify the original data, nor do they access external data sources. The rule-engine auditing trail is trivial because developers have the exact record of the data in/out of the rules.

7.6.2.1 Business Process Layer

This layer is responsible for managing the overall execution of the business process. These business processes, implemented using BPEL, can be long running, transactional,

and persistent. Process logic supports high-level transactions ("sagas") across Web service/EJB calls as well as nested sub-process transactions. The BPEL engine supports audit and instrumentation of workflow and thus is well suited for

- Separating less volatile workflow steps from more volatile business rules
- Implementing line-of-business processes
- Implementing process flows requiring compensation
- Supporting large-scale instantiation of process flows
- Designing process flows that need auditing
- Orchestrating heterogeneous technologies such as connectors, Web services, and Web Services Invocation Framework (WSIF)-enabled logic

Below list a code fragment of BPEL engine (List 7-3):

```
<Process name = "TravelAgency">
   <Sequence>
     <Receive create Instance="yes"
       Name = " receiveBookingRequest"
       portType = "bookingPT"
       Operation = "sendBookingRequest"
       Variable = "BookingRequest">
       <Source linkName = "order-to-flight"/>
     </Receive>
     <Invoke operation = "findFlights"
       inputVariable = "BookingRequest"
       outputVariable = "FlightData"
       portType = "airlinePT"
       Partner = "airline">
       <Target linkName = "order-to-flight"/>
     </Invoke>
     <! - - Invoke other service - ->
     <Flow>
       <Invoke operation = "bookCar"…/>
       <Invoke operation = "findSightSeeing-Tours/>
     </Flow>
     <Reply variable = "BookingSuggestion"
       portType = "bookingPT"
       Operation = "sendBookingSuggestion"/>
   </Sequence>
</Process>
```

List 7-3. Code Fragment in BEPL Engine

The BPEL will invoke the underlying Eligibility Web service using the code below (List 7-4):

```
<assign name="setAccount">
  <copy>
    <from variable="BPELInput" part="payload"
          query="/tns:EligibilityProcessRequest/tns:Account">
    </from>
    <to  part="parameters"
        query="/nsxml0:assessEligibility/nsxml0:Account"
        variable="webservice_account"/>
  </copy>
</assign>

<invoke name="CallEligibilityWebservice"
partnerLink="EligibilityWebservice"
       portType="nsxml0:EligibilityService" operation="
assessEligibility "
       inputVariable="webservice_account"
outputVariable="webservice_eligibilityResults"/>
```

List 7-4. Invoking Code

7.6.2.2 Web Service Layer

Service layer is a bridge between business process layer and business rules layer. It is queried by service consumers and forwards its requests to the Business Rules Broker which look up the business rule repository to get the results of the rule execution. Due to the dynamic nature of business rules, the service layer needs a flexible method to generate Web services from declarative rule descriptions. Moreover, since legacy systems and new custom components are orchestrated by BPEL process flows, issues with component compatibility, service description and data type conversion also need be addressed at this layer.

7.6.2.3 Business Rules Layer

Business rules layer is the infrastructure of enterprise information system, typically works as the home for complex logic calculation. The character of the business rule layer architecture presented in this section is separating infrastructure logic from application logic, which will result in a faster implementation of business process and business applications. To achieve this aim, a standard interface is included in this layer.

The interface works as an interpreter between clients and rule engines, hides the complicated rule implementation and execution process to end users by transforming all rule service enquires to engine recognised constructions. All rule manipulations are encapsulated in a so-called rule engine adaptor. Moreover, as regards the enterprise architecture scenario, one business task usually involves several rule engines which might be placed in different places. Therefore, a desired business rules layer should also obtain the ability of accessing various rule engines contemporarily and remotely. In general, the business rule layer should meet the following requirements:

- Provides a standard rule engine API to adopt various business rule engines integration into enterprise system.
- Allows translating between RuleML and specific rule languages required by target rule engines.
- Implements a unified interface for business rule layer communicating with web service layer.
- Supports remote rule invocation from distributed application.

In the proposed approach, a rule engine adaptor and a rule service provider are implemented in the business rule layer to accomplish these requirements. The rule engine adaptor is designed for integrating external business rule engines. The rule service provider is used for publishing rule services.

As mentioned in the BRM system description, every business rule stored in the rule repository is assigned a unique id, which is independent from rule engines. This gives great flexibility for business rule broker when invoking rules managed by a specific engine. Because the system manager only needs to identify the rule, without knowing the concrete engine the broker is using to execute the rules.

The Rule Interceptor Service is the bridge between the business rules and the executable BPEL process. The proposed approach is to intercept each incoming and outgoing BPEL Web service call to automatically apply business rules, accessible through the Business Rules Broker service. The mapping of BPEL activities to concrete

business rules is done by a mapping document, which has to be created by the BPEL designer.

The generation of the web services code from business rules is performed by a business rules based transformer component. The responsibilities of it are implementing the core rules and logic for generating, compiling and deploying web services code. The web services code is generated by using an XSLT style sheet to transform the business rules and logic to the XML code. In this business rules based transformer component, transformation algorithms provide mapping between the business rules library and XML specification. The business rules transformer has the ability to extract specific business rules for service-oriented code as XML via XSLT translation. There is a one to many correspondences between an XSLT and a business rule system. The architecture can support multiple business rules libraries with a dedicated XSLT for each system.

Below is a simple business rules transformation example. The example transforms an invoice check rule to a web service called “InvoiceRegistrationService” with one operation called “validateOrderExist”. The original business rule is showed as follows (List 7-5):

Business rule: "INVOICE-REGISTRATION"
ON (invoice) OR (e-invoice)
IF (related order exists) AND (receipt exists)
THEN begin invoice registration raise event "INVOICE-ACCEPTED"
ELSE reject invoice raise event "INVOICE-REJECTED"

List 7-5. An Example of Business Rule

Considering the example rules, the transformation from business rules to XML code can be performed by using the name of the rules file as a service name (e.g., InvoiceRegistrationService) and the rule set name as a service operation (List 7-6).

```
<? Xml version="1.0" encoding="UTF-8"?>
    <RulesTransformer>
        <RulesService name="InvoiceRegistrationService">
            <ServiceOperation name="validateOrderExist">
                <Description>
```

```
            Checking order and receipt are exist before register invoice.
            </Description>
            <Id> ordercheckrules</Id>
            <Parameter name="invoice">
            examples.Invoice
            </Parameter>
        </ServiceOperation>
    </RulesService>
</RulesTransformer>
```

List 7-6. An Example of Transformed Business Rule

Although this is a very simple example, however it reveal a standardised method for creating the web services to replace the business rules under the architecture of business rules based transformer component.

7.6.3 Workflow Integration

Traditional IS evolution approaches emphasis on determining the logical dependencies between the components in the system. These relationships specify the order of invocations (control flow) and rules for data transfer among components (data flow). According to the business process scheme, individual web services can apply on each component to provide some functionality and the greater value is then derived by combining several web services to establish more powerful applications.

Comparing to the traditional approach, the proposed approach applies BRM system as an independent component to provide decision supporting service which has advantages in improving flexibility and efficiency of IS. By separating business rules from concrete program code, each business constraint or business rule no need be integrated with the process specification and enforced throughout the process. Business rule can be invoked by any circumstance and discard from system memory after task complete. The business process modularity also helps the reusability of business rules. Moreover, with the independent BRM system, the activity of accommodating business changes can be as simple as updating the corresponding rules or add new ones. But to the process oriented approach, such situation usually results to the modification in whole process definition.

The business process is specified at an abstract level for process-oriented legacy system. Integrating business rules has to enhance legacy system with an interface to BRM system, so that the features of the latter can be used in programs written in the former. The latter relies on meta-model providing commonly used entities within a process definition, such as conditional branching, sequential and parallel activities

Take the aforementioned invoice process for example, integration is oriented to create the “virtual” data center to manage the entire distributed clinical data, while function integration is oriented to provide common functionalities for systems in healthcare institutions. None of them focus on the workflow, which is a technology that manages and monitors processes and allows the flow of work between individuals and/or departments to be defined and tracked. By implementing each system with the standard transactions, software engineers could easily achieve the workflow integration.

7.7 Summary

Service-oriented BRM systems isolate business rules from applications making the rules eminently reusable. Since business rules are organisationally universal-as opposed to application specific-rule-based engines can provide unprecedented reuse (another goal of decoupling) of business rules. In addition, business rules that are visible to non-technical users are much better understood and this transparency introduces higher accountability within teams.

Through BRMs, business rules can be changed on demand very easily. Up-to-date requirements can be implemented immediately by system users, without the need for frequent system rebuilds.

In business environment, service-oriented evolution is more like a business-oriented movement. where the work focus on how an organisation cope with the ever-changing and heterogeneous business and how they can be composed in a loosely-coupled manner to allow for flat and predictable cost of change. This is the best practice to recommend that the target services should be coarse-grained, not simply expose the fine-grained legacy assets in the same way using a new interface or middleware. Clearly, the design and development of rules, Web services, and BPEL processes involve

multiple different technologies.

Chapter 8
Case Study

Objectives

- To show how to use the proposed approach to migrate legacy systems.
- To show how to use related tools in software modernisation process.

Small examples are discussed in many chapters of this thesis. They show how functional elements of a business rules approach can be implemented. Discussing design options can provide a general appreciation of a certain number of the issues involved. Only by going to the level of program code can many of the details involved in these design decisions become apparent. Thus, it is ultimately necessary to examine the program code in the sample application to fully appreciate many aspects of the design of business rules based approach.

This chapter illustrates two application of the proposed approach in legacy system evolution by means of case studies. The work is to identify the integrated business rules and represent them as web services. The benefits from the work are greater flexibility and the ability to handle problems in terms of knowledge and symbolic reasoning.

8.1 Application for Human Resource Management

8.1.1 Introduction

The candidate enterprise employs approximately 4,000 staff, and is subdivided into a number of directorates, each being responsible for specific services. Each directorate has a resource manager who is responsible for a number of projects. The duty of the

resource manager is to fit suitably qualified people to specific jobs within each of the projects for their directorate. Project requirements and progress are monitored by Staff Management Units (SMUs) assigned to each project. It is the SMU who reports back to the resource manager within the directorate. Any vacancy that cannot be filled within the directorate is then considered across the other directorates. This involves staff being transferred between directorates, which are coordinated by the resource manager after consultation with the other directorate SMU and resource managers.

Originally, it was developed in VB 6.0 in 1999, which is a WinForm application. In 2001, the Microsoft .NET framework are released, which promised to address the old problems and meet the goals for the next-generation system. The .NET Framework is a much more consistent and object-oriented environment. New system was redeveloped in 2002 as a Web application with VB .Net 2002.

In this case study, the system will reuse existing components and an existing database. By defining and structuring their data, and identifying, extracting, and replicating the business rules contained within their legacy systems, The legacy system is rearchitected and redeployed as Web Services. The results of this experiment show that the proposed methodology and related techniques can provide much insight on the implementation of a system with a layered architecture and an efficient reengineering solution based on SOA.

8.1.2 HRMS Understanding and Partioning

The process starts from understanding the documents and source code of the legacy system. In this case study, the valuable asset of legacy system is recognised as human resource management system. The main task of HRMS was to match staff with suitable placements and the ability to hold data relating to staff skills, location, availability, personal factors, and other human resource management knowledge. HRMS is used to do job-person matching. Its process includes the consideration of vacancy criteria, skills criteria, and the preferred next work areas of staff.

According to analysis of structure of HRMS, the system skeleton can be presented as Figure 8-1.

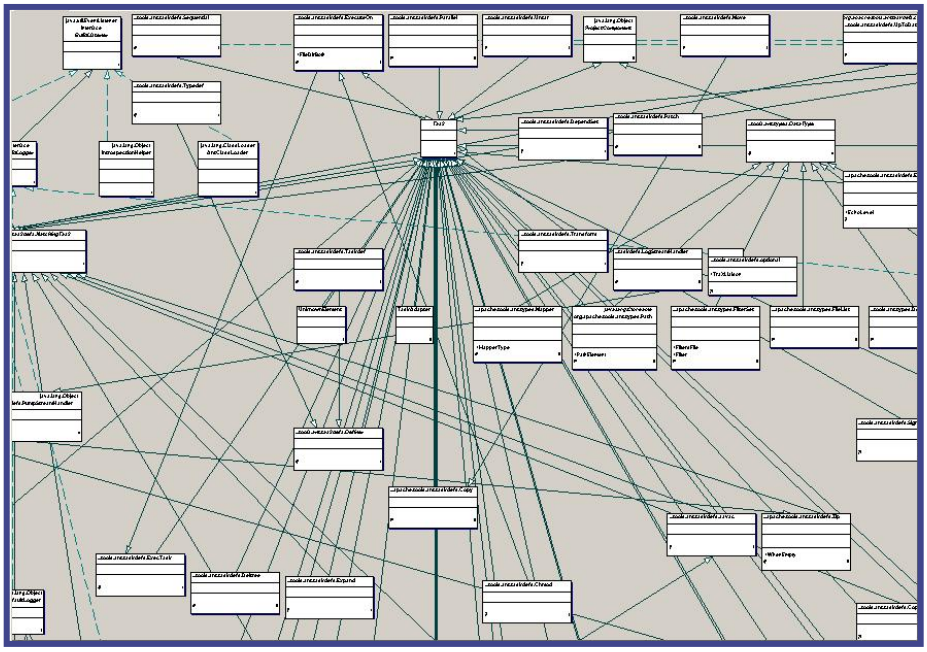

Figure 8-1. Elements Obtained by a Bottom-up Design Extraction Approach

Program slicing tools are used to understand the code. In the program slicing analysis, related classes are found, which need to be included. There are dependencies between these classes. After program slicing analysis is applied to refine the extracted code, some code refactoring activities are performed to encapsulate the legacy component.

In order to group the sliced legacy code segments and create a hierarchical structure of them, the GA-based cluster analysis is applied. The clustering analysis is carried out to identify legacy functionalities. Architects supervise the clustering analysis process and select the cutting point. The clustering method together with human supervision provides a powerful analysis on legacy systems, and represents them into modules.

8.1.3 Business Rules Extracting from HRMS

The Section demonstrates the possibility of automatic support of the business rules lifecycle by automatically generating business rules from the captured information, along with data representing domain assumptions in a case study.

Software developers can define the characteristics of each interested key word.

Table 8-1 shows a table structure used to represent the result of the knowledge acquisition from legacy system. The schema of the table includes four attributes: name, style, type, and memo. The name attribute represents the name of interested key words in the existing system. The key words will exist in three different kinds of styles, i.e., atom, rule, and variable. Atom means the word is a fact. Rule means the word represents an inference rule. Variable means the object value will be generated or supported by another event. The type attribute represents the content type of the object. Memo is to enable the developer to write down comments for the object. This will assist the developer to understand the meaning of the object during the development cycle.

Name	**Style**	**Type**	**Memo**
Find-Employee	Rule	Boolean	
First-Priority-Group	Rule	Boolean	
Skill-Sufficient	Rule	Boolean	
Display	Rule	Boolean	
Person-id	Variable	Character	
Name	Variable	Character	
Location	Variable	Character	
Preferred-Working-Area	Variable	Character	
Project-Directorate	Variable	Character	
Person-Directorate	Variable	Character	
Staff-Type	Variable	Character	
Age	Variable	Character	

Job-Required-Age	Variable	Character	
Availability	Variable	Character	
Average-Grade	Variable	Character	“high”,“middle”, or “low”

Table 8-1. The Result of the Knowledge Acquisition for the HRMS

In order to keep this example simple, only three rules are selected in the HRMS as business rules.

- The suitable employees for a project should be chosen from the first priority group. They should have sufficient skill and are willing to work in the location where the project is.
- Staff in the first priority group should be available and internally employed by company. They should be in high grade level and work in suitable directorate. In addition, their ago should meet the job requirement.
- Sufficient skill means staff’s personal skill can meet job requirements.

The execution of a job-person match begins with the matching process in vacancy criteria, which includes staff type (internal or external), location, directorate, availability, and grade. Staff whose details match vacancy details in these aspects will be selected from the database as first priority group staff for further evaluation. Staff selected from first priority group are then evaluated in their skill criteria. Each vacancy skill is used to match those of skill attainment of staff, and each skill level is checked if it meets the required level. During these processes, a skill qualification point is calculated for each staff and then evaluated to decide whether this staff will be selected as the candidate for this vacancy. The third condition is to identify if this vacancy is one of the work areas that have been recommended as the next moves of staff, and also, if this vacancy is one of preferred next work areas of staff. This information will be displayed together with result explanations for users’ reference in decision-making.

For very complex systems, once the business rules are extracted, they can be

transformed in XML and managed by BRM system.

8.1.4 Wrapping as Service

Extracted business rules and legacy components can be wrapped as Web services straightforward according to requirements. The goal of the wrapping process is to provide the component extracted from the legacy code with a WSDL interface. The technique used is to transform each entry into a method and to transform each parameter into an XML data element. The data structures will become complex elements with one or more sub-elements. The methods will have their arguments and results as references to the data element descriptions. Both the methods and the parameters will be built into an XML schema with a SOAP framework.

Within .Net platform, a wrapper can be used to expose interfaces over componentised legacy applications as well as provide meta-data descriptions of legacy data. A wrapped legacy system can function as an autonomous component and may cooperate with other components. On the basis of the analysis of the legacy application, developers decide which classes can be wrapped into Web Services and which methods can be wrapped into Web Services method according to the needs of integration. Developers should also set the properties of Web Services and Web Services method. The properties of Web Services are service name, description and namespace. The properties of Web Services method are MessageName, CacheDuration, EnableSession, TransactionOption, BufferResponse, and Description etc. List 8-1 shows a slice of generated Web Services code.

```
Imports System.Web.Services
<System.Web.Services.WebService( Description:=”SMUService”,
   Name:=”SaleBill”, NameSpace:=”URL of the Company”)>
   Public Class SaleBill
      InHerits System.Web.Services.WebService
         Public Sub New()
            MyBase.New()
         End Sub
<WebMethod(
   BufferResponse:=True, CacheDuration:=”0”, Description:=” First-Priority-Group”,
   MessageName:=”QueryBySql”,TransactionOption:=”Disabled”>
   Public Function Query(ByVal sSql As String, ByRef dt As DataSet,
                         Optional ByRef sErrDescr As String=””) As Boolean
      …
```

```
End Function
```

List 8-1. Generated Web Services for First Priority Group

8.1.5 Service Orchestration

Service integration allows the composition of services. In this case study, three rules can be applied to query the suitable candidates. Each of them can be treated as sub-services. Their execution sequences are initiated by “Service orchestration” and run by an orchestration engine. This case study is too simple to be orchestrated, but it shows that Rule based Web Service solution can simplify software development.

8.1.6 Tool Support

Tool support is an important issue for code comprehension techniques. In the case study, the main tools used are .NET Reflector and NDepend, which are introduced as the following:

.NET Reflector enables users to easily view, navigate and search through the class hierarchies of .NET assemblies even if the code for them is not available. It can decompile and analyse .NET assemblies in Visual Basic and IL. .NET Reflector helps users understand the relationships between classes and methods, find where types are instantiated and exposed, and check that the code has been correctly obfuscated before release. There are also more than 30 add-ins for .NET Reflector, which focus on different code comprehension tasks.

NDepend is used in this case study for dependency management. NDepend is a tool that simplifies managing a complex .NET code base. It help users analyse code structure, specify design rules, plan massive refactoring, do effective code reviews and master evolution by comparing different versions of the code. NDepend supports the Code Query Language (CQL) for maximum flexibility. Basically, NDepend considers the code base as a database and some CQL statements can be used to query and check some assertions on this database. NDepend also supports more than 80 code metrics, which are related to code organisation, code quality, code structure, code coverage and so on.

NDepend also can detect dependency cycles and help users remove them.

Developers also experience two open source business rules engines: Jess and Drools. Since the case study is very simple, they did not use them to to make inferences.

8.2 Application for Customer Resource Management

8.2.1 Introduction

The candidate online shop offers a large variety of products through different channels to customers with diverse needs. The lack of a formal commitment and the ease with which shoppers can prefer competitors make the process of building a loyal customer base tougher for retailers. Quality of commodities and competitive pricing are often not enough to stand out from the competition

In such a dynamic environment, the success of a retailer is heavily weighted to the quality of its customer care. A competitive retailer should try to understand its customers by gaining insight into their needs, attitudes, and behaviors. Ideally each customer should be treated as an individual and the enterprise should operate on a one-to-one basis. Since this approach is obviously not possible, an efficient alternative is to divide the customers into groups with different characteristics and apply different business rules that best address their specific features. As mentioned in previous chapters, different business rules can be developed according to the specific business objectives which are embedded in the legacy system of the organisation. As a result, the retailer is committed to deploying best practices in account management and customer care operations to achieve rapid customer response and fast issue resolution.

To realise these goals, the online shop needs a customer support and account management software to automate these best-in-class service delivery processes. It was imperative that the solution could:

- Support collaborative issue resolution: Many incoming support incidents are complex and require the customer care team to work collaboratively with engineering, services and account managers to resolve issues. Many of these

individuals are geographically dispersed across the country. As a result, the online shop requires the software system provide an infrastructure for real-time collaboration and communication.

- Facilitate fast knowledge transfer at shift change: The customer care engineers work in a shift-oriented 24x7 environment. At the end of the shift, they need to transfer detailed case knowledge to the incoming shift engineers.

- Enable account management processes: The system should implement an account-centric support and escalation model providing account managers with visibility into all issues associated with their accounts.

- Seamless knowledge transfer: The online shop has deployed a platform to proactively monitor and manage its customers. The new support system should should make use of their previous knowledge

8.2.2 Legacy Database Analysis

Customer Resource Management systems (CRMs) include two kinds of knowledge: customer information and the strategy to handle various customers. The first step is to analyse customer information and rearrange them for business rules implementation.

Customer information contains entire customer incident lifecycle from information capture, relevant team assembly and collaborative resolution to incident close. They all stored in Microsoft Access database. The advantage of Access is that it is a stand-alone programming language and database, and it is reasonably easy to follow program code implemented in it. That is why it has been chosen for the online shop application. Figure 8-2 shows the screen of the working Access database.

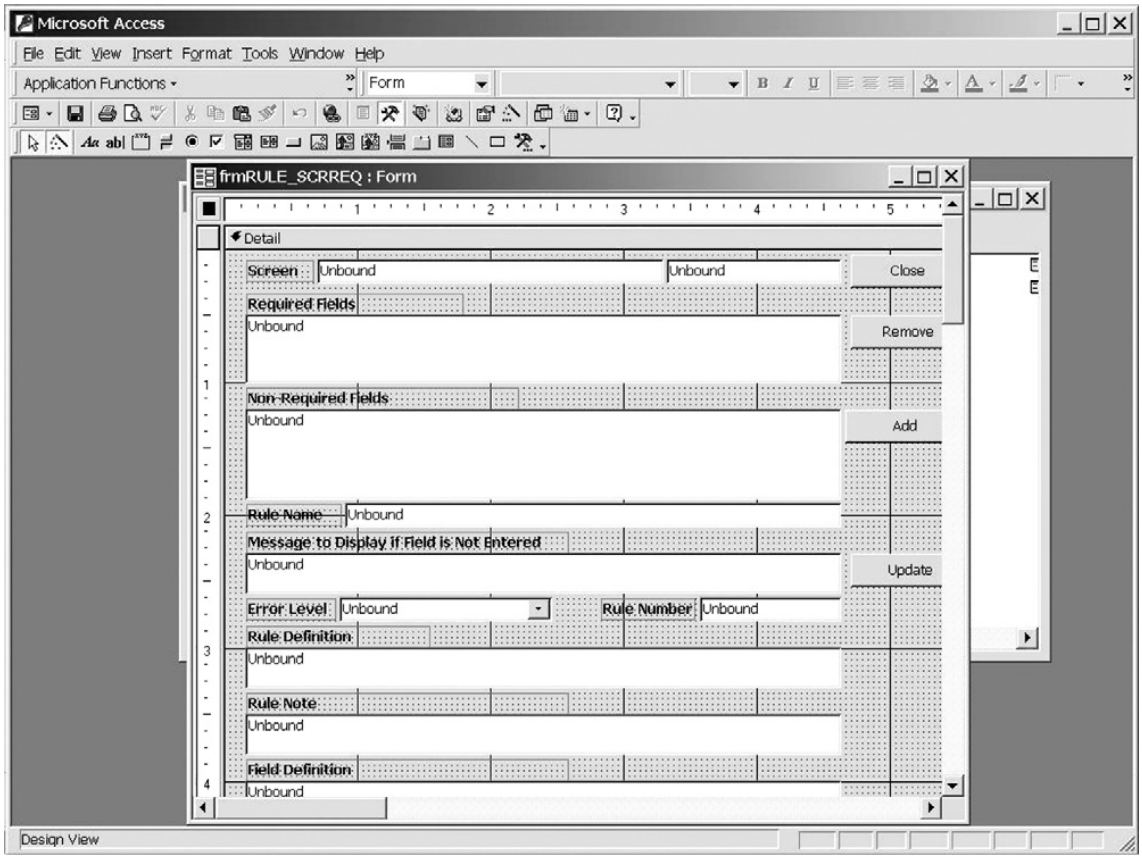

Figure 8-2. A Screen of the Access Database

There are many tables in the sample database. Those that are needed for business rules functionality are discussed in the Section 8.2.4. Some tables which record the customer information are presented at various places in the database to make specific points about special customers. However, not all of these tables are actively used by the legacy CRM system. They only marked as an investment partnership to enable database user can identify them when needed. Those that are used in this way are centered around the Investment table. Figures 8-3 and 8-4 present the logical and physical structures of the tables that hold business data and that are used for customer categorisation.

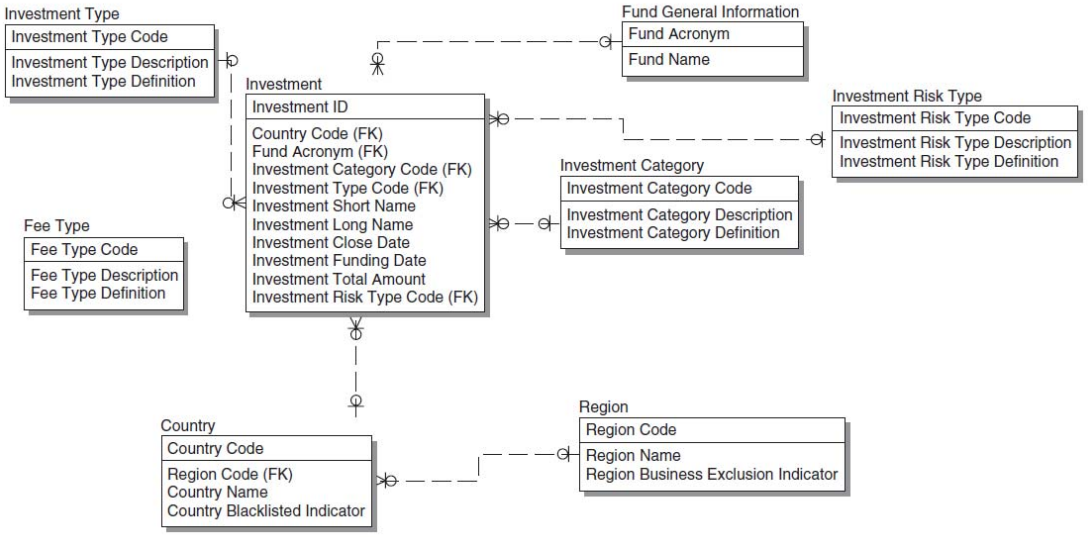

Figure 8-3. Logical View of Business-Relevant Database Tables

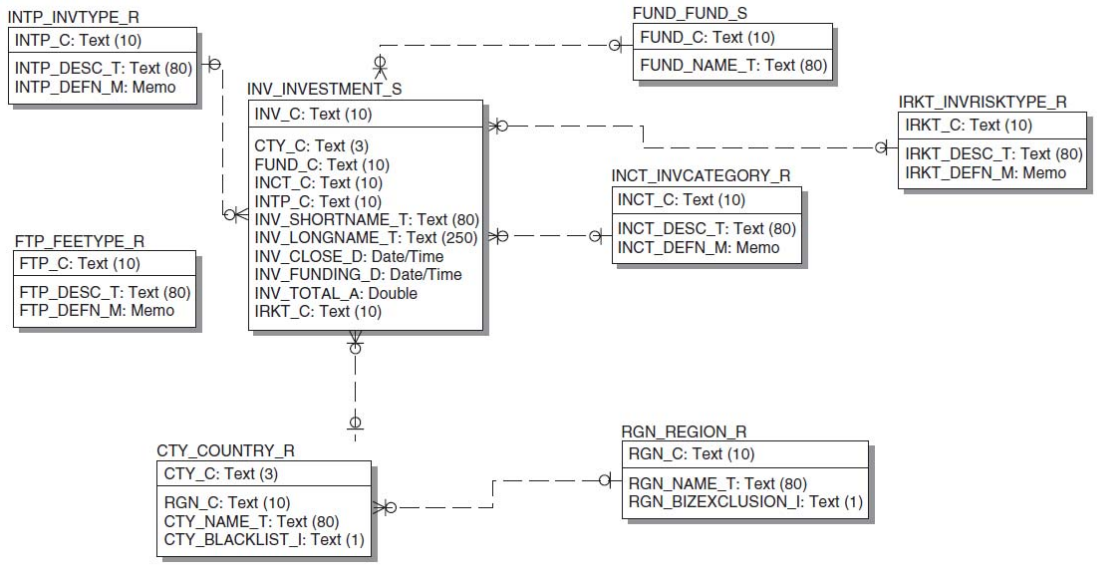

Figure 8-4. Physical View of Business-Relevant Database Tables

Comprehensive customer resource knowledge captured during the resolution process. At this stage, a recency, frequency, monetary (RFM) analysis method is implemented to examine and group customers according to their purchase frequency, recency, and value. The construction of the RFM indicators is a simple data management task which does not involve any data mining modeling. It does, however,

involve a series of aggregations and simple computations that transform the raw purchase records into meaningful scores. In order to perform RFM analysis each transaction should be linked with a specific customer (ID) so that the purchase history of that customer can be tracked and investigated over time. Fortunately, in most situations, the use of a loyalty program makes the collection of personalised transactional data possible. Table 8-2 lists the result draw from RFM analysis.

Superstars	• The most loyal customers • Highest value • Highest frequency • High spending on private labels
Golden customers	• Second highest value • High frequency • Average spending on private labels
Typical customers	• Average value and frequency • Average spending on private labels
“Exceptional occasions” customers	• The second lowest frequency after “Dormant customers” • Large basket • Low recency values (long time since their last visit)
“Everyday” shoppers	• Increased frequency of transactions • Small basket • Private labels • Medium to low value
Dormant customers	• Lowest frequency and value • Long time since their last visit (lowest recency values)

Table 8-2. Customer Category

8.2.3 Identifying Legacy Code

Once customers be classified, the next step is extracting business rules from legacy

code. The old CRM system used by online shop was a web-based legacy system developed by asp.net. System users manage customers information by openning asp web pages which are connect to Access database as data sources. The operation interface of the legacy system is shown in Figure 8-5.

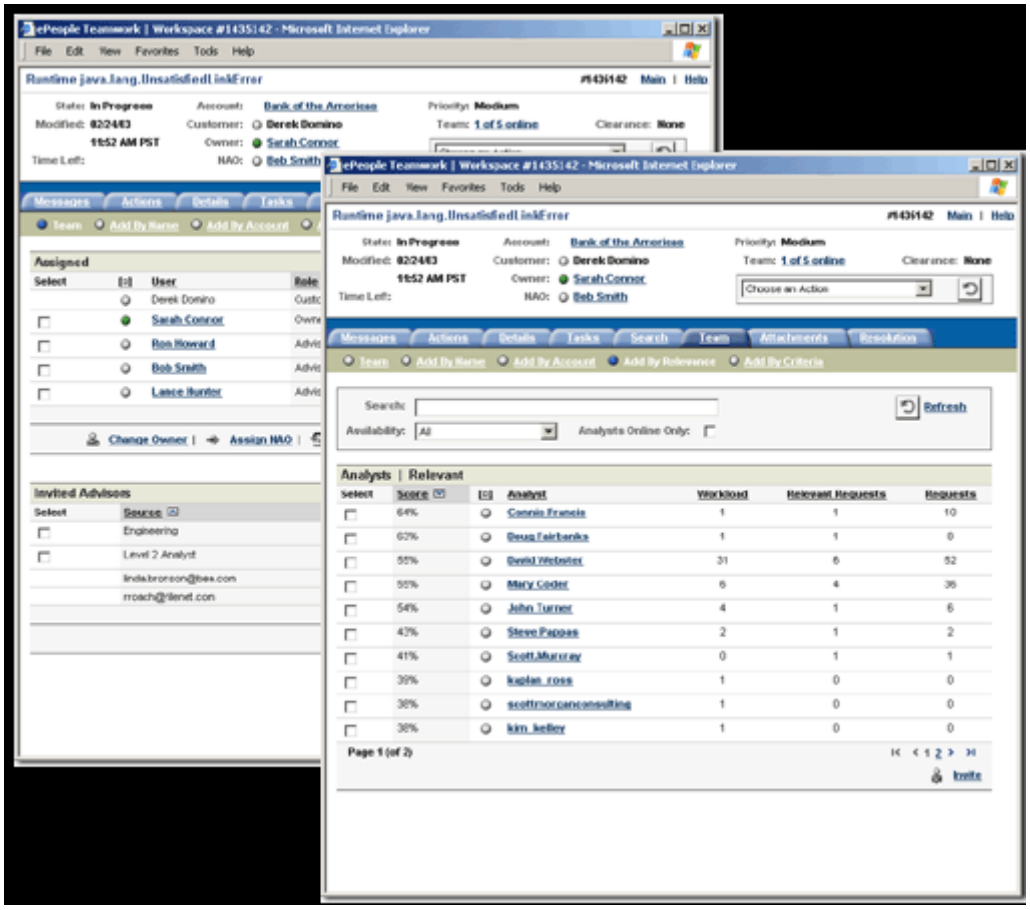

Figure 8-5. Screen Shot of the Web-base Legacy System Interface

The web-based character makes the business rules extraction work very simple. Because the asp code is mainly used for representation purpose. Software engineers can easily identify the description code and concentrate on the very limited variables handleing codes. Moreover, there still have much business logic embedded in the database. This logic is derived from the decisions made for the activities of each customer. It can not be obtained by using technical method, but only can be found by the business professionals. The business experience plays a very important role in this part of business rules.

8.2.4 Storing Busienss Rules in Rule Repository

The rule entity is the core elements in the business rules approach. Rule entity includes various properties listed in Table 8-3.

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique ID for the rule
	Version	Text	Version number, in whatever format suits
	LastModifiedDate	Date/Time	When the rule was created or last changed
	LastModifiedBy	Text	Name of the person responsible
	Type	Number	Link to type of rule table (see Table 8-4)
	Identifier	Text	Application-independent identifier for the rule
	DescriptiveName	Text	Longer textual description of rule
	Status	Text	Indicator to show the current status of the rule
	Notes	Memo	Field for more extensive free-format notes
	RuleStatement	Memo	The business rule statement (in XML)
	StatementTemplate	Number	Link to template table (see Table 8-5)
	FormalStatement	Memo	A formal representation of the rule
	BusinessCategory	Number	Link to a business category table (see Table 8-9)

Table 8-3 Rule Entity Table

In the rule entity table, there are several key points link with the information from external tables. The modification history is traced using “Last Modified Date” and “Last Modified By” properties. These two properties link to other tables to provide more modification information. Rules are classified by their types. The “Type” property

indicates which group the specific rule should belong to. The “Identifier” property stands with the rule itself. It marks the rule in particular situation. One rule can have aliases as identifiers in various execution situations. The “Status” property of the rule entity will have text values such as *approved*, *pending* or *ineffective*. This property can be extended to an external table which will store a fixed set of status values, and the status text field can be replaced by a link to the new table. Rule statements are expressed in the form of an XML string in a memo field to avoid arbitrary size constraints, which preserves the inherent structure but can be processed to provide a straightforward natural language rendition. The structures of rule entities are abide by one of the templates from template table and linked through “Statement Template” property. Same as the “Type” property mentioned above, the business rules can also be allocated to one of the categories according to their business purposes using the “Business Category” property.

The following tables illustrate every external table intrinsic to the rule shown in Table 8-3.

	Field Name	Data Type	Description
*	Rule ID	Number	Link to rule
*	EnforcementType	Text	Link to enforcement type
	FromDate	Date/Time	Rule enforcement starts date
	ToDate	Date/Time	Rule enforcement ends date

Table 8-4 Rule Enforcement Table

	Field Name	Data Type	Description
*	Type	Text	Name of the enforcement type
	Description	Memo	Longer description of the nature of the enforcement

Table 8-5 Rule Enforcement Type Table

Table 8-4 and 8-5 stored all the information related with rule enforcement process. One thing should be noted is the rule implementing period. Business rules may be applicable through out the whole system runtime process or may apply only for a particular period. The former type of rules reflecting in the table 8-5 is no records in the “FromDate” and “FromDate” field.

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique identifier for the rule type
	TypeName	Text	The textual name of the type
	Description	Memo	Expanded description of the type

Table 8-6 Rule Type Table

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique identifier for the category
	CategoryName	Text	Human-readable name of the category
	Description	Memo	Expanded description

Table 8-7 Business Category Table

Rules are stored in repository where they are categorised by various types of property. The *Property Type* is used to model the imported properties from business event, system processing information and data processing information. Property types at the business level are business processes, business entities, business events and business goals. This level allows rules to be allocated to one of a number of potential business categories with business meaning, such as core business, legislation, productivity, regulation, and security. At information system level, property types correspond to information system modeling elements, which include system functions, data entities, and system events. At the lower data processing levels, they correspond to software system processes, classes and hardware components. Properties are actual system model elements and rules which can be linked to make the required set. The control of

assignments will be improved if the allowed property types for every abstraction level are recorded.

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique identifier for the rule template
	Name	Text	Human-readable name of the template
	Structure	Memo	The template, expressed in XML

Table 8-8 Rule Template Table

Rules correspond to rule templates in the repository. The rule template prescribes the language and syntax of rule statements in accordance with particular patterns. In many circumstance, a rule might be applicable to several data. In such cases, rule templates allow for the creation of rules with empty slots to be filled in later. Through the template supporting, rule repository provides a mean for automatic rule structure dissemination and transformation. The structuring capabilities of XML can also help rule templates to be stored in a Text field.

	Field Name	Data Type	Description
*	Rule ID	Number	Link to rule
*	AuthorityID	Number	Link to responsible Authority
	AuthorityType	Text	The responsibility assumed by Authority
	AuthorisationReference	Text	Reference to supporting documentation
	AuthorisationDate	Date/Time	When the responsibility came into force

Table 8-9 Rule Authority Table

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique Identifier for authority

	RuleOfAuthority	Text	The person involved
	PositionOdAuthority	Text	The position of person in the organisation

Table 8-10 Authority

	Field Name	Data Type	Description
*	Type	Text	Name of the type of authority
	Description	Memo	Expended description of authority type

Table 8-11 Authority Type

Security is one of the most important aspects of enterprise systems. As the basis system component, the rule repository must have a detailed authorisation discipline link every rule with authority types. Various kinds of authorities are involved with rules: the expert who was the original source for the definition, the business owner who has to sign off the rule, the project manager who deems it in scope, and so on. All can be accommodated by these tables, along with a link to the documentation that supports their authority and the date that this was effective. See Tables 8-9, 8-10 and 8-11.

The above tables outline the basic design of business rules stored in the rule repository. It can be easily extended to adopt any additional localised features.

Business rules working with others to take effects. The relationships among them provide a means for rule traceability from the origin to implementation. Rule repository supplies various ways of cross-referencing rules. According to the difference of implementing condition, rule repository can add more references to meet customised requirements. The basic cross-reference structure is shown in Figure 8-6.

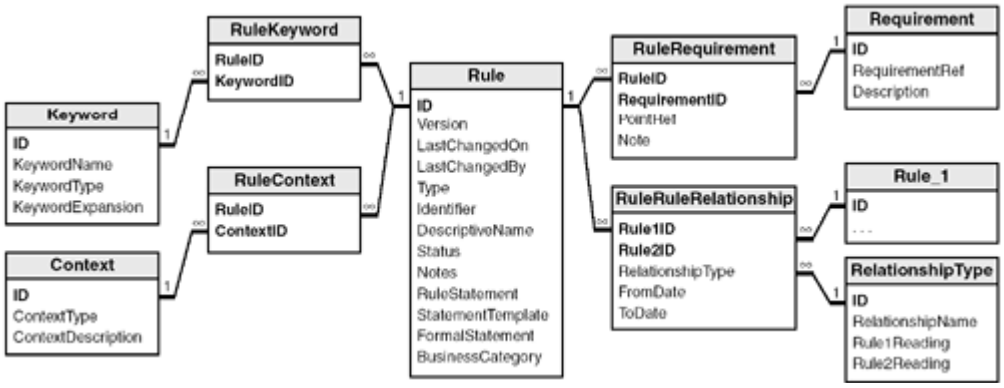

Figure 8-6. Cross-Reference Structure

	Field Name	Data Type	Description
*	RuleID	Number	Link to rule
*	KeywordID	Number	Link to keyword

Table 8-12 Rule Keyword Table

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique identifier for the keyword
	KeywordName	Text	Human-readable name of the keyword
	KeywordType	Text	Classification of keyword usage
	KeywordExpansion	Memo	Optional expansion of keyword

Table 8-13 Keyword

Keyword searching is the most common way of tagging rules with specific terms to the analysis. For example, saw a rule that prohibited students from receiving a loan; that rule might be tagged with the keywords “student” and “loan”.

	Field Name	Data Type	Description
*	RuleID	Number	Link to rule

*	ContextID	Number	Link to context

Table 8-14 Rule Context Table

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique identifier for the context
	ContextType	Text	Human-readable context type
	ContextDescription	Memo	Further detail on the context

Table 8-15 Context Table

It is useful to be able to discuss a group of rules together, perhaps in a review session. The context may be somewhat ad-hoc, such as “the rules associated with the customer service use cases”, and a particular rule may be considered in several contexts. See Tables 8-14 and 8-15.

	Field Name	Data Type	Description
*	Rule1ID	Number	Link to rule 1
	Rule2ID	Number	Link to rule 2
	RelationshipType	Number	Link to relationship type
	FromDate	Date/Time	When the relationship become effective
	ToDate	Date/Time	When the relationship expire

Table 8-16 Rule Relationship

	Field Name	Data Type	Description
*	ID	AutoNumber	Unique identifier for the relationship type
	RelationshipName	Text	Textual name of the type of relationship

	Rule1Reading	Text	The way the relationship looks from rule 1
	Rule2Reading	Text	The way the relationship looks from rule 2

Table 8-17 Relationship Type

An important way for rules to represent business policy and strategy is procedures. Some of the knowledge is distinctly procedural. For example, the tax liability calculation work can not be done unless the income and expenditure has been known. In such case, rule will relate to one another. A good rule repository should offer the ability for rulesets to invoke related rules to execute and return values.

In Figure 8-6, the rule entity is corresponds to rule1 and rule2 respectively. Their relationships are kept in Table 8-16 and 8-17. The two readings of the relationship provide the information to explain the perspectives of each rule. For example, in an exception relationship, “rule1 (has an exception) rule2” and “rule2 (is an exception to) rule1”. The Table 8-16 and 8-17, as they stand, supports the linking of only two rules in one single way. If a need for multiple relationships emerged, the extension would be straightforward.

8.2.5 Developing Web Service

Business rules are loaded from a variety of resources such as asp code and database. Considering this situation, developers designed a java based rule engine package which including two java classes to collect and choose specifically rules as the execution sets for BRM systems. One class which called Rule Loader Set is used for loading business rules from rule database of system. A Rule Filter Set then call Rule Administrator Class to executes these rules and evaluates execution result. The Rule Filter Set regenerating a list of rule objects by querying the structural metadata of a rule execution set and output the rule objects list to the next transformation process.

```
// Get an input stream to a Rule Engine Package.
   // The Rule Loader set begin.
   InputStream inStream =
      RuleloaderSet.class.getResourceAsStream(
```

```
      "/home/res/rule/general.lib");
   System.out.println("Acquired InputStream to RuleFilterSet:
"+ inStream);
// Parse the rules from the rule list document.
   RuleFilterSet res1 =
      RuleAdministrator.class.getRuleExecutionSet(null).cre
   ateRuleExecutionSet( inStream, null );
      inStream.close();
      System.out.println("Loaded RuleExecutionSet: " + res1);
```

List 8-2. Load a Set of Rules from InputStream File

Above List 8-2 is an example of loading a set of rules from an InputStream file. This example demonstrates the life cycle of rules selection using the rule engine package.

Rules engines such as JRules provide tools to generate wrapper Web services or Session Beans to wrap a newly developed ruleset. Web services developers will be instrumental in creating a wrapper to expose ruleset as a Web service.

XML is a key standard for integrate rules, EJBs, BPEL process flows, and Web services. BPEL uses XML natively to access data while Web services use it to serialise data (and will use it unmodified in Doc/Literal invocations). XML can be used directly in rules. By marshalling, XML can be transformed directly into a JAXB object tree. Rules can be executed with native Java Objects.

The following List 8-3 is an example of business rule from business rule repository

```
<?xml version="1.0" encoding="utf-8" ?>
<Rules desc="Demonstation of Rules Processing Engine"
author="Jeff Bramlett">
  <Rule id="SetupRules">
   <Action type="run" actionId="SetupValues">
      <Match />
      <Assign />
```

```
   </Action>
  </Rule>
  <Rule id="SetupValues">
     <Actions>
        <Action type="makeVar" actionId="Qty">
           <Match />
           <Assign>0</Assign>
        </Action>
        <Action type="makeVar" actionId="Weight">
           <Match />
           <Assign>100</Assign>
        </Action>
        <Action type="makeVar" actionId="Price">
           <Match />
           <Assign>100</Assign>
        </Action>
     </Action>
  </Rule>
  <Rule id="CalculateInvoice">
  <Rule id="CalculateWeights">
  <Rule id="SetDiscounts">
     <Action type="var" actionId="StarQtyDiscount">
        <Match>
        <![CDATA[ StarQty > 50 ]]>
        </Match>
        <Assign>.3</Assign>
     </Action>
  </Rule>
  <Rule id="CalculateTotalPricing">
     <Actions>
        <Action type="var" actionId="TotalPrice">
           <Match />
```

```
            <Assign>(StarQty*StarPrice)-((StarQty*StarPrice)
         * StarQtyDiscount)
            </Assign>
         </Action>
         <Action type="var" actionId="ShippingTotal">
            <Match />
            <Assign>ShippingRate * WeightTotal</Assign>
         </Action>
         <Action type="var" actionId="InvoiceTotal">
            <Match />
            <Assign>(PriceSubTotal -
         CustomerTypeDiscountAmount)+ SalesTaxTotal +
         ShippingTotal
            </Assign>
         </Action>
      </Actions>
   </Rule>
   <Triggers>
      <Trigger id="TriggerOne" varID="InvoiceTotal"
   ruleID="TriggerTest" />
      </Triggers>
</Rules>
```

List 8-3. An Example of XML Business Rule

8.2.6 Tool Support

The process of this case study mainly used RFM analysis method and Java.

The RFM analysis is a common approach for understanding customer purchase behavior. It is quite popular, especially in the retail industry. As its name implies, it involves the calculation and the examination of three KPIs - recency, frequency, and monetary - that summarise the corresponding dimensions of the customer relationship

with the organisation. The recency measurement indicates the time since the last purchase transaction of the customer. Frequency denotes the number and rate of purchase transactions. Monetary indicates the value of the purchases. These indicators are typically calculated at a customer level through simple data processing of the available transactional data.

Different from the first case study, the second case study uses Java to expose business rule function as web service is for a better integration with other applications. The development of a java based rule engine package can be performed under the JavaBean environment.

8.3 Summary

In the case studies, service-oriented reengineering process is performed. The first phase is gathering the system information from source code and other documentation. Then, reusable components are indentified and extracted by slicing techniques and GAs-based Clustering Analysis. Business rules are extracted and isolated. Finally, the exterlised as Web services.

The end result is that developing new applications is faster than before and IT is now focusing on what is important to the organisation, namely the rules for running the business. By this case study, the following points are summarised:

The target Web service can be defined and designed properly, which simplifies the reengineering process.

The Web service consumers communicate with SOSR Web services via the standard Web service interface, so the implementation for this in the Web service consumer side is not difficult.

Tool support is very important. Powerful tools can help both software engineer and end users.

Chapter 9
Conclusions

This chapter concludes the thesis by summarising the work presented and assessing the extent to which the research aims and objectives set out in Chapter 1 have been addressed. First a summary of the thesis is presented, then the significance of the contributions and the limitations issues are discussed, and finally directions for future work are suggested.

9.1 Summary of Thesis

Legacy systems built using earlier software-development paradigms focused almost exclusively on fixed business processes. As the business environment changes, functional enhancements are inevitable in the evolution of any successful software. To have any hope of building an information system that meets new business requirements, this thesis uses both static and dynamic analysis to describe a multifaceted information structure that is used to capture the requirements.

The overall goal of this thesis is to align business with information technology. And to achieve this target, this thesis provides a step-by-step practical approach for evolving legacy systems using a business rules approach.

Business rules are only one element in the legacy system evolution. To understand how they fit into the big picture, software engineers also have to look at other aspects of a system. The way in which relate rules to all the other elements is through one or more business and software models that describe the system from various perspectives. Even if either of them been fully understood, software engineers need to understand business models to a certain extent so that rules can be seen in their proper context.

Within a well-balanced analysis model, business rules are used to represent the

logic that lies behind business decisions. But rules do not stand alone; they must be integrated along with representations of other key business features, such as:

- Events, either internal or external, that may impact business activities
- Processes: sequences of tasks that are carried out by the business
- Objects, either real or abstract, that the business needs to deal with and the interrelationships between these objects
- The policies and goals of the business
- Actors and roles: the entities, both human and machine, with which the business wants to interact
- The organisational units that go to make up the business
- Narratives that describe scenes in the life of the business

In this thesis research in software evolution and in particular in the area of business rules is presented. The intention of this thesis is to provide business and IT professionals with a good understanding of the field of business rules. Armed with this knowledge, business users should be able to make informed decisions about the planning and execution of information systems projects, including answering such questions as: How do we identify and record business rules? Where and how should we implement them? Should we build or buy any related technology? What management issues are involved?

Coupled with the web service, the business rule based legacy system evolution is believed to be able to create a class of services that is flexible and relatively easy to maintain, upgrade, and integrate.

9.2 Significance of Contributions and Evaluation

The work presented in this thesis contributes to the development of Web service ontologies and, implicitly, to the realisation of the semantic Web services idea. The contributions fall in three major categories: The main contributions of this thesis are in

three areas：

1. **Reducing the workload of analysing legacy system.**

The methodological framework addresses the drawbacks of many contemporary techniques. In particular, it is designed to combine bottom-up reverse engineering approaches with top-down business process engineering methods, introducing a meet-in-the-middle approach that is capable of aligning new business processes with legacy systems. This alignment is facilitated by a sophisticated process of comparison and adaptation to ensure the processes and systems fit.

2. **Developing an approaches to extract the embedded business knowledge and integrat them in SOA.**

The web service technique has offered business users an unprecedented opportunity to have the same access to information as their sighted counterparts. However, the ability of IT professionals and business users to operate SOA sysetms has introduced an inequality. This thesis makes a significant contribution to redress this imbalance by providing a set of techniques and principles such that isolate the embedded business rules and represent them to business users in a natural language.

3. **Helping reengineering work to succeed by making non-technical business users involve into the system evolution work.**

The alignment of business processes with information technology has often been studied from a rather abstract vantage point that is believed to provide insufficient guidance to reverse engineer and information system developers. It is the main contribution of the methodological framework presented in this thesis to offer a body of pragmatic methods and techniques to assist in aligning business processes and legacy systems.

9.3 Conclusion and Future Directions

A business rules approach to systems development promises to be the most

practical and desirable way to evolve legacy systems. It can help software engineers build better, easily changeable systems faster than any previous approach.

It is now time for a step-by-step approach for applying the theory to technology in the world of e-business, Web-based, object-oriented, and relational database systems development. This thesis combines best practice systems design principles plus adherence to the principles of a business rules approach, thereby making the work of legacy system evolution is a easier, faster, and cheaper task.

Even though this chapter concludes this thesis, in fact it does not provide an “end” to the research but it provides a “beginning” to further research. One of the grand challenges surrounding legacy system evolution is to make it a continuous evolving process. Implementing legacy business knowledge should not be implemented as a one-time project with a fixed end but should ideally be realised as a process that is performed periodically or even constantly to guarantee business knowledge is fit for information technology. This thesis provides encouraging results to realise this by solving the problems that discourage reverse engineers from accessing legacy knowledge. The work presented in this thesis will provide a firm foundation for ongoing research into supporting better information system performance for nontechnical users.

References

[1] "Rule Enabling Applications with Oracle Business Rules," Oracle, 2002.

[2] "Java Rule Engine API™ JSR-94," BEA Systems, Inc., 2003.

[3] "Web Services architecture overview," IBM, Software Group, 2000.

[4] M. Aiello, C. Platzer, F. Rosenberg, H. Tran, M. Vasko, and S. Dustdar, "Web Service Indexing for Efficient Retrieval and Composition," in *Proceedings of the 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services*, 2006, pp. 63-63.

[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services," BEA, IBM, Microsoft, SAP AG and Siebel Systems 5 May 2003.

[6] J. Ash, L. Chung, K. D. Souza, W. S. Lai, H. V. d. Linde, and Yung Yu, "AT&T's MPLS OAM Architecture, Experience and Evolution," in *IEEE Communications Magazine*. vol. 42, 2004, pp. 100-111.

[7] H. Berkowitz, *Building Service Provider Networks*, 1st ed.: John Wiley & Sons, 2002.

[8] P. J. Bernhard, G. Faulkner, and O. Frieder, "XML Strategies for Legacy Database Search," in *IT Professional*. vol. 5, 2003, pp. 16-21.

[9] M. J. A. Berry and G. S. Linoff, *Data Mining Techniques For Marketing, Sales, and Customer Relationship Management*: Wiley Publishing, 1997.

[10] H. Boley, B. Grosof, and S. Tabet, "RuleML Tutorial." vol. 200: The RuleML Initiative, 2005.

[11] F. Botto, *Dictionary of e-Business*, 2nd ed. Chichester: John Wiley & Sons, 2003.

[12] D. M. Brandon, *Software Engineering for Modern Web Applications: Methodologies and Technologies*. London: Information Science Reference, 2008.

[13] D. Breitigam and W. Division, "A Reusable, Event-based Architecture for Business Rule Models," in *Proceedings of the 22nd Digital Avionics Systems Conference*, 2003, pp. 3.A.2 - 3.1-8.

[14] R. Butleris and K. Kapocius, "The Business Rules Repository for Information Systems Design," in *Proceedings of the Research Communications of the 6th East European Conference*, 2002, pp. 64-77.

[15] B. CA, M. Barai, and V. Caselli, *Service Oriented Architecture with Java*. Birmingham: Packt, 2008.

[16] F. Cabitza, M. Sarini, and B. D. Seno, "D Jess - A Context-Sharing Middleware to Deploy Distributed Inference Systems in Pervasive Computing Domains," in *Proceedings of the International Conference on Pervasive Services*, 2005, pp. 229-238.

[17] L. Cai and Y. Fei, "Business Process Reengineering in Early Stages of Corporation Life Cycle," in *Proceedings of the Portland International Conference on Management of Engineering and Technology*, 1999, pp. 533-534.

[18] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "A Flexible Wrapper For The Migration Of Interactive Legacy System to Web Services," in *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, 2006.

[19] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating Interactive Legacy Systems To Web Services," in *Proceedings of the Conference on Software Maintenance and Reengineering*, 2006.

[20] L. Chambers, *The Practical Handbook of Genetic Algorithms*, 2nd ed. Florida:

CRC Press, 2001.

[21] W. Y. Chang, *Network-centeric Service-oriented Enterprise*. Dordrecht: Springer, 2007.

[22] D. A. Chappell, *Enterprise Service Bus*. Sebastopol, CA: O'Reilly, 2004.

[23] A. Charfi and M. Mezini, "Hybrid Web Service Composition: Business Processes Meet Business Rules," in *Proceedings of the 2nd International Conference on Service Oriented Computing*, 2004.

[24] F. Chen, S. Li, H. Yang, C.-H. Wang, and W. C.-C. Chu, "Feature Analysis for Service-Oriented Reengineering," in *Proceedings of the 12th Asia-PacificSoftware Engineering Conference*, 2005.

[25] H. K. Cheng, Q. C. Tang, and J. L. Zhao, "Web Services and Service-Oriented Application Provisioning: An Analytical Study of Application Service Strategies," *IEEE Transactions on Engineering Management,* vol. 53, pp. 520-533, 2006.

[26] M. Chisholm, *How to Build a Business Rules Engine*. San Francisco: Elsevier, 2004.

[27] D. K. W. Chiu, W. C. W. Chan, G. K. W. Lam, S. C. Cheung, and F. T. Luk, "An Event Driven Approach to Customer Relationship Management in e-Brokerage Industry," in *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2002.

[28] D. N. Chorafas, *Enterprise Architecture and New Generation Information Systems*. Florida: St. Lucie Press, 2002.

[29] M. A. Cibrán, M. D'Hondt, and V. Jonckers, "Aspect-Oriented Programming for Connecting Business Rules," in *Proceedings of the 6th International Conference on Business Information Systems*, 2003.

[30] D. Cohen, G. Larson, D. McDougal, and B. Ware, "Extending Life Cycle of

Legacy Systems," in *Proceedings of the International Conference on Computer Networks and Mobile Computing*, 2003, pp. 291-296

[31] F. Cohen, *FastSOA*. San Francisco: Elsevier, 2007.

[32] C. Collet, G. Vargas-Solar, and H. Grazziotin-Ribeiro, "Open Active Services for Data-Intensive Distributed Applications," in *Proceedings of the International Database Engineering and Applications Symposium* 2000, pp. 349-359.

[33] M. C. Daconta, L. J. Obrst, and K. T. Smith, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*: John Wiley & Sons, 2003.

[34] B. Daum and U. Merten, *System Architecture with XML*, 1st ed. San Francisco: Morgan Kaufmann, 2002.

[35] A. Delves, "Using XML: A PHP Developer's Primer." PHPBuilder.com, 2007.

[36] S. Demeyer, S. Ducasse, and O. Nierstrasz, *Object-Oriented Reengineering Patterns*. San Francisco: Morgan Kaufmann, 2002.

[37] M. Diouf, S. Maabout, and K. Musumbu, "Standard Business Rules Language: why and how," in *Proceedings of the International Conference on Artificial Intelligence*, 2006.

[38] D. Doval, S. Mancoridis, and B. S. Mitchell, "Automatic Clustering of Software Systems using a Genetic Algorithm," in *Proceedings of the Software Technology and Engineering Practice*, 1999, pp. 73-81.

[39] D. Drislane, "Business Rules Modeled: A primer on modeling business rules and inference networks using Rational Rose.," BeggsHeidt Enterprise Consulting, Inc. 2003.

[40] L. Èeponis and O. Vasilecas, "Logical View of Architecture of Business Rules Repository," *Informacijos Mokslai,* vol. 36, pp. 139-147, 2006.

[41] E. Eilam, *Reversing: Secrets of Reverse Engineering*. Indianapolis: Wiley

Publishing, 2005.

[42] H.-E. Eriksson and M. Penker, *Business Modeling with UML: Business Patterns at Work*, 1st ed.: John Wiley & Sons, 2000.

[43] T. Erl, *Service-Oriented Architecture A Field Guide to Integrating XML and Web Services*: Prentice Hall, 2004.

[44] J. Ferguson, D. Kaminsky, J. Larsen, L. Miras, and W. Pearce, *Reverse Engineering Code with IDA Pro*. Burlington: Syngress, 2008.

[45] J. Footen and J. Faust, *The Service-Oriented Media Enterprise*. Burlington: Elsevier, 2008.

[46] E. Friedman-Hill, *Jess in Action: Rule-Based Systems in Java*: Manning Publications, 2003.

[47] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison Wesley, 2000.

[48] J. Garland and R. Anthony, *Large-Scale Software Architecture*. Chichester: John Wiley & Sons, 2003.

[49] H. Gaur and M. Zirn, *BPEL Cookbook*. Birmingham: Packt, 2006.

[50] M. Gen and R. Cheng, *Genetic Algorthms and Engineering Oprimzation*. New York: John Wiley & Sons, 2000.

[51] P. C. M. Gottgtroy and M. d. P. B. Gottgtroy, "Significant Conceptual Modelling - A Health Care Enterprise Case Study," in *Proceedings of the 12th International Workshop on Database and Expert Systems Applications:*, 2001, pp. 917-921.

[52] I. Graham, *Business Rules Management and Service Oriented Architecture A Pattern Language*. Chichester: John Wiley & Sons Publishing , 2007.

[53] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R.

Neyama, *Building Web Services with Java™: Making Sense of XML, SOAP, WSDL, and UDDI*: Sams Publishing, 2001.

[54] M. Grand, *Java™ Enterprise Design Patterns: Patterns in JavaTM* vol. 3: John Wiley & Sons, 2001.

[55] B. N. Grosof, "Standardizing XML Rules: Preliminary Outline of Invited Talk," in *the IJCAI-01 Workshop on E-business and the Intelligent Web*, 2001.

[56] A. Groznik and A. Kovacic, "Business Renovation: From Business Process Modelling to Information System Modelling," in *Proceedings of the 24th International Conference on Information Technology Interfaces*, 2002, pp. 405-409.

[57] H. Guo, C. Guo, F. Chen, and H. Yang, "Wrapping Client-Server Application to Web Services for Internet Computing," in *Proceedings of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2005, pp. 366-370.

[58] E. Gutentag and A. Gregory, "XML-based Rules," in *Proceedings of the Extreme Markup Languages Conference*, 2001, pp. 73-84.

[59] R. P. Hall, "Seven Ways to Cut Software Maintenance Costs," in *Techniques of Program & System Maintenance (QED)* Parikh: Information Sciences Inc., 1988.

[60] B. v. Halle, *Business Rules Applied*. New York: John Wiley & Sons, 2002.

[61] A. E. Hassan and R. C. Holt, "Architecture Recovery of Web Applications," in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 349 - 359.

[62] C. Heinrich and B. Betts, *Adapt or Die: Transforming Your Supply Chain into an Adaptive Business Network*: John Wiley & Sons, 2003.

[63] P. Herzum and O. Sims, *Business Component Factory: A Comprehensive*

Overview of Component-Based Development for the Enterprise: John Wiley & Sons, 2000.

[64] I. Ho, Z. Komiya, B. Pham, H. Kobayashi, and K. Yana, "An Efficient Framework for Business Software Development," in *Proceedings of the International Conference on Cyberworlds*, 2003, pp. 336-343.

[65] M. A. Hoffman and D. L. Carver, "Reverse Engineering Data Requirements," in *Proceedings of the IEEE Aerospace Applications Conference*, 1996, pp. 269-277.

[66] T. Hu, S. Zhang, and B. Li, "P/spl I.bar/DROCE Model based Supply Chain Management System and Its Verification," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2003, pp. 4085-4090.

[67] J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, and K. P. Sycara, *Service-Oriented Computing: Agents, Semantics, and Engineering*. Berlin: Springer, 2007.

[68] M. Hugos, *Essentials of Supply Chain Management*, 1st ed.: John Wiley & Sons, 2003.

[69] I. Jacobson and F. Lindstriim, "Re-engineering of old systems to an object-oriented architecture.," *ACM SIGPLAN Notices,* vol. 26, pp. 340 - 350, November 1991.

[70] L. C. Jain, V. Palade, and D. Srinivasan, *Advances in Evolutionary Computing for System Design*. Berlin: Springer, 2007.

[71] S. Jarzabek, *Effective Software Maintenance and Evolution*. New York: Auerbach Publications, 2007.

[72] S. Jarzabek, *Effective Software Maintenance and Evolution*. New York: Auerbach Publications, 2007.

[73] J. Jeng, D. Flaxer, and S. Kapoor., "RuleBAM: A Rule-Based Framework for Business Activity Management," in *Proceedings of the IEEE International Conference on Services Computing*, 2004, pp. 262-270.

[74] J.-J. Jeng, H. Chang, and J.-Y. Chung, "A Policy Framework for Business Activity Management," in *Proceedings of the IEEE International Conference on E-Commerce*, 2003, pp. 238-245.

[75] K. A. D. Jong, *Evolutionary Computation*. Cambridge: The MIT Press, 2008.

[76] L. Juszczyk, A. Michlmayr, and C. Platzer, "Large Scale Web Service Discovery and Composition using High Performance In-Memory Indexing," in *Proceedings of the 9th IEEE International Conference on E-Commerce Technology*, 2007, pp. 509-512.

[77] V. Kapsalis, K. Charatsis, M. Georgoudakis, and G. Papadopoulosl, "Architecture for Web-based services integration," in *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society*, 2003, pp. 866-871.

[78] A. A. P. Kazem and S. Lotfi, "An Evolutionary Approach for Partitioning Weighted Module Dependency Graphs," in *Proceedings of the 4th International Conference on Innovations in Information Technology*, 2007, pp. 252-256.

[79] K. M. Khan and Y. Zheng, *Managing Corporate Information Systems: Evolution and Maintenance*. London: Idea Group, 2005.

[80] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston: Pearson Education, 2006.

[81] A. Kovacic and A. Groznik, "The Business Rule-Transformation Approach," in *Proceedings of the 26th International Conference on Information Technology Interfaces*, 2004, pp. 113-117.

[82] K. E. Kurbel, *The Making of Information Systems*. Berlin: Springer-Verlag, 2008.

[83] W. Lamersdorf, V. Tschammer, and S. Amarger, *Building the E-Service Society*. Boston: Springer Science, 2004.

[84] J. C. S. d. P. Leite and P. M. Cerqueira, "Recovering Business Rules from Structured Analysis Specifications," in *Proceedings of the 2nd Working Conference on Reverse Engineering*, 1995, pp. 13-21.

[85] H. Li, M. Xue, and Y. Ying, "A Web-based and Integrated Hospital Information System," in *IDEAS Workshop on Medical Information Systems: The Digital Hospital*, 2004, pp. 157-162.

[86] L. Lin, S. Embury, and B. Warboys, "Business Rule Evolution and Measures of Business Rule Evolution," in *Proceedings of the 6th International Workshop on Principles of Software Evolution*, 2003, pp. 121-125.

[87] L. Lin, S. M. Embury, and B. C. Warboys, "Facilitating the Implementation and Evolution of Business Rules," in *Proceedings of the 21st IEEE International Conference on Software Maintenance*, 2005, pp. 609-612.

[88] M. H. Linehan and D. F. Ferguson, "Business Rule Standards -- Interoperability and Portability," IBM Software Group 2005.

[89] D. S. Linthicum, *Enterprise Application Integration*: Addison Wesley, 1999.

[90] C.-H. Lung and J. E. Urban, "An Approach to the Classification of Domain Models in Support of Analogical Reuse," *ACM SIGSOFT Software Engineering Notes,* vol. 20, pp. 169-178, August 1995.

[91] Z. Maamar, D. Benslimane, and A. Anderson, "Using Policies to Manage Composite Web Services," in *IT Professional*. vol. 8, 2006, pp. 47-51.

[92] N. H. Madhavji, J. C. Fern´andez-Ramil, and D. E. Perry, *Software Evolution and Feedback*. Chichester: John Wiley & Sons, 2006.

[93] O. Maimon and L. Rokach, *Decomposition Methodology for Knowledge Discovery and Data Mining*. Singapore: World Scientific, 2005.

[94] B. Maizlish and R. Handler, *IT Portfolio Management Step-by-Step*. New York: John Wiley & Sons, 2005.

[95] R. C. Martin, "UML Class Diagrams," Object Mentor 1997.

[96] T. Mattern and D. Woods, *Enterprise SOA: Designing IT for Business Innovation*: O'Reilly, 2006.

[97] K. McDonald, A. Wilmsmeier, D. C. Dixon, and W. H. Inmon, *Mastering the SAP Business Information Warehouse*, 1st ed.: Wiley Publishing, 2002.

[98] J. McGovern, S. Tyagi, M. Stevens, and S. Matthew, *Java Web Services Architecture*: Morgan Kaufmann, 2003.

[99] P. J. Meilir, *The Practical Guide to Structured Systems Design*, 2 ed.: Prentice Hall PTR, 1988.

[100] J. Meng, S. Y. W. Su, H. Lam, and A. Helal, "Achieving Dynamic Inter-Organizational Workflow Management by Integrating Business Processes, Events and Rules," in *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.

[101] T. Mens and S. Demeyer, *Software Evolution*. Berlin: Springer-Verlag, 2008.

[102] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Berlin: Springer-Verlag, 1996.

[103] H. Mill, A. Mill, S. Yacoub, and E. Addy, *Reuse based Software Engineering*. New York: John Witey & Sons, 2001.

[104] D. E. Millard, Y. Howard, S. Chennupati, H. C. Davis, E.-R. Jam, L. Gilbert, and G. B. Wills, "Design Patterns for Wrapping Similar Legacy Systems with Common Service Interfaces," in *Proceedings of the 4th European Conference on Web Services*, 2006, pp. 191-200.

[105] B. S. Mitchell and S. Mancoridis, "Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements," in

Proceedings of the IEEE International Conference on Software Maintenance, 2001, pp. 744 - 753.

[106] T. Morgan, *Business Rules and Information: Aligning IT with Business Goals* 1st ed.: Addison-Wesley, 2002.

[107] T. Morgan, *Business Rules and Information Systems: Aligning IT with Business Goals*. Indianapolis: Addison Wesley, 2002.

[108] C. Nagl, F. Rosenberg, and S. Dustdar, "VIDRE – A Distributed Service-Oriented Business Rule Engine based on RuleML," in *the 10th IEEE International Enterprise Distributed Object Computing Conference*, 2006, pp. 35-44.

[109] C.-K. Nam and J.-H. J. Bae, "A framework for processing active documents," in *The 6th Russian-Korean International Symposium on Science and Technology*, 2002, pp. 122-125.

[110] J. Oberleitner, F. Rosenberg, and S. Dustdar, "A Lightweight Model-Driven Orchestration Engine for e-Services," in *the 6th VLDB Workshop on Technologies for E-Services* 2005, pp. 48-57.

[111] B. Orriens, J. Yang, and M. P. Papazoglou, "A Framework For Business Rule Driven Web Service Composition," in *Conceptual Modeling for Novel Application Domains*. vol. 2814/2003: Springer Berlin / Heidelberg, 2003, pp. 52-64.

[112] A. Paschke and M. Bichler, "SLA Representation, Management and Enforcement," in *the IEEE International Conference one-Technology, e-Commerce and e-Service*, 2005, pp. 158-163.

[113] M. D. Penta, G. Casazza, G. Antoniol, and E. Merlo, "Modeling Web Maintenance Centers Through Queue Models," in *the 5th European Conference on Software Maintenance and Reengineering*, 2001, pp. 131-138.

[114] A. Perkins, "Business Rules = Meta-Data," in *34th International Conference on*

Technology of Object-Oriented Languages and Systems, 2000, pp. 285-294

[115] M. Pinzger, M. Fischer, H. Gall, and M. Jazayeri, "Revealer: A Lexical Pattern Matcher for Architecture Recovery," in *Proceedings of the 9th Working Conference on Reverse Engineering, 2002. Proceedings.* , 2002, pp. 170- 178.

[116] S. G. Powell and R. J. Batt, *Modeling for Insight*. New Jersey: John Wiley & Sons, 2008.

[117] I. Redbooks, *Patterns: Implementing an SOA Using an Enterprise Service Bus*, 1st ed.: International Business Machines Corporation, 2004.

[118] R. Richards, *Pro PHP XML and Web Services*: Apress, 2006.

[119] D. Rosca, S. Greenspan, and C. Wild, "Enterprise Modeling and Decision-Support for Automating the Business Rules Lifecycle," *Automated Software Engineering,* vol. 9, pp. 361-404, 2002.

[120] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*. Indianapolis: Wiley Publishing, 2008.

[121] F. Rosenberg and S. Dustdar, "Business Rules Integration in BPEL–A Service-oriented Approach," in *Proceedings of the 7th IEEE International Conference on E-Commerce Technology*, 2005, pp. 476-479.

[122] F. Rosenberg and S. Dustdar, "Design and Implementation of a Service-Oriented Business Rules Broker," in *Proceedings of the 7th IEEE International Conference on E-Commerce Technology Workshops*, 2005, pp. 55-63.

[123] F. Rosenberg and S. Dustdar, "Towards a Distributed Service-Oriented Business Rules System," in *Proceedings of the 3rd European Conference on Web Services*, 2005.

[124] F. Rosenberg, C. Nagl, and S. Dustdar, "Applying Distributed Business Rules-The VIDRE Approach," in *Proceedings of the IEEE International*

Conference on Services Computing, 2006, pp. 471-478.

[125] R. G. Ross, "The Business Rule Approach," *Computer,* vol. 36, pp. 85-87, 2003.

[126] R. G. Ross, *Principles of the Business Rule Approach*: Addison Wesley, 2003.

[127] I. Rouvellou, K. Rasmus, and D. Ehnebuske, "Extending Business Objects with Business Rules," in *Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages*, 2000, pp. 238-249.

[128] S.N.Sivanandam and S.N.Deepa, *Introduction to Genetic Algorithms*. Berlin: Springer-Verlag, 2008.

[129] P. Saengpongpaew and M. D. Smitasiri, "Business Competitive Advantage through Advanced Planning Simulator," in *Proceedings of the IEEE International Engineering Management Conference*, 2003, pp. 56-60.

[130] J. Sametinger, *Software Engineering with Reusable Components*. Berlin: Springer-Verlag, 1997.

[131] K. Sartipi, "Software Architecture Recovery based on Pattern Matching," in *Proceedings of the International Conference on Software Maintenance*, 2003, pp. 293 - 296.

[132] V. Shaiva, "Designing Adaptive Components for a Services Oriented Architecture," in *Proceedings of the International Conference on Information Technology: Research and Education*, 2003, pp. 390-394.

[133] T. C. Shan, "Building a Service-oriented eBanking Platform," in *Proceedings of the IEEE International Conference on Services Computing*, 2004, pp. 237-244.

[134] S. J. Shepard, "Policy-Based Networks: Hype and Hope," in *IT Professional*. vol. 2, 2001, pp. 12-16

[135] M. G. Shields, *E-Business and ERP: Transforming the Enterprise* John Wiley & Sons, 2001.

[136] R. Stackowiak, J. Rayman, and R. Greenwald, *Oracle Data Warehousing and Business Intelligence Solutions*: Wiley Publishing, 2007.

[137] J. Strassner, "DEN-ng: Achieving Business-Driven Network Management," in *IEEE/IFIP Network Operations and Management Symposium*, 2002, pp. 753-766.

[138] N. C. Suazo and J. O. O. Aguirre, "Aspect-Oriented Web Services Orchestration," in *Proceedings of the 2nd International Conference on Electrical and Electronics Engineering and XI Conference on Electrical Engineering*, pp. 72-76.

[139] D. Sun, K. Wong, and D. Moise, "Lessons Learned in Web Site Architectures for Public Utilities," in *Proceedings of the 5th IEEE International Workshop on Web Site Evolution*, 2003, pp. 93-100.

[140] A. Talevski, E. Chang, and T. S. Dillon, "Reconfigurable Web Service Integration in the Extended Logistics Enterprise," in *IEEE Transactions on Industrial Informatics*. vol. 1, 2005, pp. 74-84.

[141] P. Tonella and A. Potrich, *Reverse Engineering of Object Oriented Code*: Springer, 2005.

[142] T.-M. Tsai, H.-K. Yu, H.-T. Shih, P.-Y. Liao, R.-D. Yang, and S.-c. T. Chou, "Ontology-Mediated Integration of Intranet Web Services," *Computer,* vol. 36, pp. 63-71, 2003.

[143] K. Umapathy and S. Purao, "Designing Enterprise Solutions with Web Services and Integration Patterns," in *Proceedings of the IEEE International Conference on Services Computing*, 2006, pp. 111-118.

[144] O. Vasilecas and A. Smaizys, "Business rule based data analysis for decision support and automation," in *Proceedings of the International Conference on Computer Systems and Technologies*, 2006, pp. II.9-1-II.9-8.

[145] Y. Vasiliev, *SOA and WS-BPEL*. Birmingham: Packt, 2007.

[146] M. Wang and H. Wang, "Intelligent Agent Supported Business Process Management," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 71b-71b.

[147] X. Wang, J. Sun, X. Yang, Z. He, and S. R. Maddineni, "Application of Information-flow Relations Algorithm on Extracting Business Rules from Legacy Code," in *Proceedings of the 5th World Congress on Intelligent Control and Automation*, 2004, pp. 3055-3058

[148] X. Wang, J. Sun, X. Yang, Z. He, and S. R. Maddineni, "Automatically Identifying Domain Variables based on Data Dependence Graph," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2004, pp. 3389-3394.

[149] M. Wermelinger, G. Koutsoukos, R. Avillez, J. Gouveia, L. Andrade, and J. L. Fiadeiro, "Using Coordination Contracts for Flexible Adaptation to Changing Business Rules," in *Proceedings of the 6th International Workshop on Principles of Software Evolution*, 2003, pp. 115-120.

[150] L. Wilkes and R. Veryard, "Service-Oriented Architecture: Considerations for Agile Systems," in *Microsoft Architect Journal*, 2004.

[151] I. Wong-Bushby, R. Egan, and C. Isaacson, "A Case Study in SOA and Re-Architecture at Company ABC," in *Proceedings of the 39th Hawaii International Conference on System Sciences*, 2006, pp. 179b - 179b.

[152] R. Xu, I. Donald and C. Wunsch, *Clustering*. New Jersey: John Wiley & Sons, 2009.

[153] L. Yan, R. Wolniewicz, and R. Dodier, "Customer behavior prediction - it's all in the timing," *IEEE Potentials,* vol. 23, pp. 20-25, Oct-Nov 2004.

[154] H. Yang, *Software Evolution with UML and XML*. London, Idea Group, 2005.

[155] H. Yang and M. Ward, *Successful Evolution of Software Systems*. London: Artech House, 2003.

[156] E. Yourdon, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*: Prentice Hall, 1979.

[157] Z. Zhang, H. Yang, and W. C. Chu, "Extracting Reusable Object-Oriented Legacy Code Segments with Combined Formal Concept Analysis and Slicing Techniques for Service Integration," in *Proceedings of the 6th International Conference on Quality Software*, 2006, pp. 385-39